PhD Thesis

Inter-Organisational Intrusion Detection System Communication to implement Network Defence

Michael Pilgermann

A submission presented in partial fulfilment of the
requirements of the University of Glamorgan / Prifysgol Morgannwg
for the degree of Doctor of Philosophy

August 2006

Supervised by *Andrew Blyth* and *Gaius Mulley*

Certificate of Research

This is to certify that, except where specific reference is made, the work described in this thesis is the result of the candidate. Neither this thesis, nor any part of it, has been presented, or is currently submitted, in candidature for any degree at any other University.

Candidate: _____

Director of Studies: _____

Date: _____

Abstract

Intrusion Detection System (IDS) technology has increasingly been deployed in recent computer networks as part of the security measures in order to combat the ever increasing number of threats, inter-connected computers are facing these days. One of the major improvements that have been applied to these intrusion detection systems, is the introduction of distribution features, allowing the components to exchange and correlate audit data and, this way, act as units instead of isolated solutions.

This PhD thesis describes a novel concept called Inter-Organisational Intrusion Detection System (IOIDS) technology, which pushes the idea of distributing audit data from intrusion detection system to a new level. By introducing trust relationships between parties and mapping them into the system, IOIDS provides a way to exchange IDS audit data across organisational boundaries. Thereby, commercial confidentiality can be maintained by employing state-of-the-art security technologies and strict distribution of all features.

The analysis of the results for this project proves that an implementation of communication channels for IDS audit data between organisations is possible. The integration of ids audit data from machines located at different geographical sides results in a clear distribution towards nowadays security situation. The time consuming process of manual exchange of this kind of information, as currently performed, can be tightened significantly by automating this process.

Acknowledgements

First of all, I thank my director of studies, Dr. Andrew Blyth, for his great supervision and support throughout the entire project time of my PhD. He has not only guided me through the process in a very helpful manner, consulted me in technical questions and been an exceptional partner for many project discussions, but has also been a contact person for any concern around the research.

A number of persons have supported me with my project, often in form of numerous discussions or any other kind of help. Just to name a few from this list I had liked to thank Stilianos Vidalis, Vivienne Mee, Evangelis Morakis, Konstantinos Xynos and Jonathan Corcoran. In the end has the entire Information Security Research Group of the School of Computing provided a supportive and friendly atmosphere for carrying out my research.

How could one finish a PhD thesis without being backed up by their family and friends? I thank my mother, Christine Pilgermann, my brothers, the rest of the family and many friends here in Cardiff and back home in Germany for supporting me mentally, cheering me up and just always being there for me whenever I needed them.

Contents

List of Figures

List of Tables

Listings

Chapter 1.

Introduction

In recent decades the evolution of the Internet has been marked by steady, rapid growth. Corporations started to provide internet accessible services and more and more workstations have been connected to the Internet. Soon, a corresponding increase of attacks against networks has been realised. (Cert/CC (2004)) For the protection of networks besides other security mechanisms such as Firewalls and Anti-Virus software, Intrusion Detection Systems (IDS) technology has been deployed in order to identify attacks in a contemporary way and enable security staff to react in a prompt and adequate manner.

For today's deployments, however, data captured at several locations in commercials, universities and governmental institutions has only been correlated inside these organisations. Consequently, each corporation acts in an isolated way and has to be attacked itself or the knowledge has to be updated manually in order to become aware of a new security thread.

The purpose of the Inter-Organisational Intrusion Detection System (IOIDS) infrastructure has been the research of opportunities and technologies for establishing a secure and trustworthy communication channel between organisations for automatic querying, validation and integration of security relevant information from disparate organisational Intrusion Detection Systems located at different geographical sides.

1.1. Hypothesis

Intrusion Detection Systems located in separate organisations can be made to communicate and share information via a XML based peer-to-peer architecture in a secure and non-reputable manner, while maintaining commercial confidentiality, allowing the network to react as a single entity to a computer network attack (CNA).

1.2. Contribution to Science

The Inter-Organisational Intrusion Detection System infrastructure describes a novel approach for information sharing across organisational boundaries by establishing trust relationships between parties and base sharing and integration decisions on those ones. This way, it clearly puts a number of features on top of existing approaches, which mainly focus on the structuring, normalisation and generalisation of available audit data.

With IOIDS a communication platform has been designed and evaluated, which exchanges knowledge in a reliable and secure manner by utilising state-of-the-art technology for grid systems, public key infrastructures and peer-to-peer technology. Standards for information exchange in the intrusion detection system context have been researched and an XML based communication application has been put into place. The introduction of policy based access control mechanisms allows the local node to protect its information and share it only with desired parties.

The idea of Inter-Organisational Intrusion Detection has been acknowledged by the academic society as a novelty, which is expressed by the number of documents published around the topic as part of the project. (see section 8.1.1 for details)

1.3. Structure

The rest of the thesis is structured the following way:

Chapter 2 provides an overview about the state of the art intrusion detection systems and their subcategory distributed intrusion detection system technology. Besides this core technology, other related approaches such as grid technology, message exchange formats and security related topics, which are also important for the project, are briefly discussed. An introduction to information security in general puts the whole project in its overall context.

Chapter 3 draws attention to background information for the project and, this way, presents some potential practical deployment scenarios for the projects as well as defines the objectives for the project on a high and abstract level. Last but not least, a terminology section introduces a common set of vocabulary to be used for the remaining chapters of the thesis.

Following that, the thesis lays down the borders for the project and, in this way, describes the scope of the research for this thesis. Within chapter 4, firstly, an overview is given about shortcomings of current available approaches for distributed intrusion detection. Afterwards, the experiment for IOIDS is described in three parts, namely the theoretical comparison of features by evaluation of available documentation for similar approaches, a high-level overview about the design of the proof-of-concept architecture and, last but not least, the

detailed definition of the analysis process, which is able to evaluate the project outcome.

Chapters 5 and 6 provide the technical details for the experiment; detailing the information about the design and implementation of the IOIDS application. As the Inter-Organisational Intrusion Detection System is coming in two major components, namely the subjacent communication platform Grid for Digital Security (G4DS) and the actual Inter-Organisational Intrusion Detection System application IOIDS, each of them is being addressed by a single chapter. Each of the two chapters has been divided into the two parts: Design and implementation; the former one addressing a more general kind of information for the component in question, the latter one drawing attention to implementation details. Interaction between the two components has been addressed in detail as part of the IOIDS explanations in section 6.3.1.

Following the technical information about the architecture of the components the project evaluation is discussed in detail in chapter 7. As laid down in the experiment definition beforehand in chapter 4, the analysis is carried out in two parts; namely a feature comparison of IOIDS with other approaches in the DIDS context and a practical experiment part carried out in a laboratory environment in order to compare features of the Inter-Organisational Intrusion Detection System with the ones of similar or related approaches. Results from both parts are compiled and a structured overview of the outcomes mirrors the IOIDS features against the predefined objectives for the project.

The last chapter in the thesis brings together the information from all parts of the thesis and puts the outcome of the project back into the broader context. Apart from an overview of the achieved results clear statements are presented, giving information about the contributions to science, the IOIDS project could make. Last but not least, a list has been assembled, which shows opportunities for taking the outcome of this project further for future developments.

Chapter 2.

State of the Art

2.1. Introduction

In the beginning for the IOIDS project, background information had to be researched and state of the art related fields and technologies had to be examined. Besides the core technology of intrusion detection, there were also some other technologies to be addressed as the project was gaining outcome from them.

Consequently, this chapter is addressing the following issues in the order given by this list:

- An introduction to security in information technology places intrusion detection system technology into its context.
- An overview about intrusion detection systems provides information about the concepts of IDS in general.
- Another section focuses solely on distribution features for intrusion detection systems technology and presents its evolution by providing an exhaustive list within this research area.
- Section 2.4.5 stresses state of the art technologies available for establishing secured connections between parties.
- An overview is provided about the standards available for exchanging security related information in a structured manner.
- As the project benefits from research outcomes from grid technology, an overview about the history and state of the art for this technology is provided as well.
- An introduction into data persistence explains briefly the differences in current available database technologies.

2.2. Security in General

Since organisations have become increasingly aware of the importance of their network infrastructures, as well as the threats faced by them a lot of money, time and effort have been put on the development of strategies, guidelines, programs and sensitisation processes to protect this sensitive and important bit of the organisation's assets. The most popular ones may be categorised as follows:

- Development of overall security concepts, addressing all possible threats as well as all components to be protected for organisations
- Threat assessment in order to assemble a priority list for correct facing the threat problem
- Systematic penetration testing in order to check the effectiveness of the aforementioned security concepts
- Development of guidelines for sensitisation of users for the threat of information violation (including for example the adequate choice of passwords and a protected location for their storage)
- Centralisation and standardisation of the network management
- Development of projects in a technical manner in order to facilitate aforementioned organisational actions, which include:
  - Anti Virus Software for the protection against viruses, worms and trojans
  - Firewall systems at gateways between networks as well as personal firewalls on local machines
  - Intrusion Detection Systems with their sub-categories Distributed Intrusion Detection Systems and Intrusion Prevention Systems (IPS)
  - Penetration testing tools such as port scanners or vulnerability detectors

2.3. Intrusion Detection - one technology in Information Security

Before drawing attention to Intrusion Detection the expression *intrusion* itself needs to be defined: "A Network Intrusion is any unwanted or unauthorised action being taken across the network that affects remote resources." (Heberlein et al. (1992)); or as described in (Crosbie

and Spafford (1994)): "An Intrusion is any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource."

Examples of network intrusions are (Heberlein et al. (1992)):

- Unauthorised modifications of system files that permit unauthorised access to either system or user information
- Unauthorised access to user file space
- Unauthorised modifications of user files / information
- Unauthorised modifications of tables or other system information in network components
- Unauthorised use of computing resources (perhaps through the creation of unauthorised accounts or through the unauthorised use of existing accounts)

Intrusion Detection instead may be described the following way: "Intrusion Detection is defined to be the problem of identifying individuals [or threat agents] that are using a computer system without authorisation (i.e. crackers) and those who have legitimate access to the system but are exceeding their privileges (i.e. insider threat)." (Balasubramaniyan et al. (1998); Mukherjee et al. (1994); Snapp et al. (1991a))

The fundamental idea of Intrusion Detection was invented by Anderson with his technical report in 1980 (Anderson (1980)). In the early years of this technology Intrusion Detection was limited to network intrusion detection. The first occurrence of a grown up approach was in 1987 with the well known paper of Denning, which is thought to be the foundation for Intrusion Detection Systems (Denning (1987)).

Over the years, many changes have since been made in the area of Intrusion Detection Systems with their sub-categories. The efficiency of these systems was challenged due to the overwhelming amount of data they produced; more and more efforts have been put on the developments in the areas of data correlation, data abstraction and data merging rather than simply gathering as much information as possible (Morakis et al. (2003)).

2.3.1. Types of Intrusion Detection Systems

Early developments for intrusion detection system technology were simply based on network intrusion detection only. Nowadays, intrusion detection covers a broader collection of technologies. The following classification is commonly used for grouping those ones (NSS (2002); Pilgermann (2003)):

Network Intrusion Detection (NIDS) – Most mature intrusion detection system technology. Sensors capture network traffic and are responsible for observing an entire network segment.

Host Intrusion Detection (HIDS) – Performs intrusion detection functionality on a single host only, usually on nodes that require special protection such as central servers, gateways or RAS Servers. It processes information from the host like system event log data, file integrity or CPU and memory utilisation in order to identify malicious activities.

Hybrid Intrusion Detection – This technology is also called Network Node Intrusion Detection and it integrates the two technologies NIDS and HIDS for a single node. Literally, it means that a host intrusion detection system is extended by network intrusion detection capabilities in order to enable it to identify a wide set of attack patterns.

Network Intrusion Prevention (NIPS)) – Network Intrusion Prevention Systems are looking to overcome the drawback of Network Intrusion Detection System based on their reactive nature. In contrast to NIDS these systems carry out their work in a proactive manner: NIPS are usually deployed as in-line nodes on gateways between network segments in order to capture and evaluate traffic. Only if the traffic is declared as being non-malicious the data is passed on to its final destination.

Host Intrusion Prevention (HIPS) – Similar to NIPS do Host Intrusion Prevention systems carry out their work in a proactive manner on a single host. The most popular technology for this purpose is called System Call Interception, in which each system call on the host in question is intercepted by the IPS and checked against a pre-defined policy for this host.

2.3.2. Detection technologies

Furthermore, detection mechanisms were improved and this way, several approaches besides the two original mechanisms, namely misuse detection and anomaly detection (Mukherjee et al. (1994)) have been established, although some of them are just derived approaches. Basically, we distinguish the following five detection methodologies (Pilgermann (2003); NSS (2002)):

1. Pattern matching: Most popular and most mature detection methodology, which attempts to detect malicious activities by searching for predefined patterns inside the network traffic

2. Stateful pattern matching: Extended version of pattern matching, which looks to overcome the problems with fragmented attacks by tracing the network traffic and reassembling fragmented pieces of network traffic

3. Protocol decoding: Reassembles the network traffic and, with the knowledge about specifications of different network protocols, it is able to identify malicious behaviour inside the traffic

4. Heuristic analysis: Utilises a kind of algorithmic logic, which the alarm decisions are based on. Often statistical evaluations of a special traffic type are used for these algorithms

5. Anomaly analysis: Instead of searching for malicious behaviour the normal behaviour is being defined (using a learning phase or based on the setting of parameters) and significant deviations from this behaviour are reported to be malicious behaviour

Each of these methodologies comes with advantages and disadvantages regarding the detection rate, false positive[1] rate, processing time and management efforts; though it's commonly agreed that only an adequate combination of both leads to a satisfactory protection of the infrastructure.

2.4. Distributed Intrusion Detection

Quite recently, a development in the area of Intrusion Detection technology may be recognised, namely its extension by distribution features. With former approaches the roles of the components in an Intrusion Detection topology were quite clearly separated the following way:

- Gathering components (sensors): Gathering the data, possibly pre-processing of the data into a format which can be processed by the processing component

- Processing component (detectors): Analysing and correlating of data received from several sensors

- Analysing components (reporter): Alerting in different kinds of communication channels (such as email, SMS, alert window) and post-processing the proceeded data for different kinds of reports

[1] False positives are alerts, which are raised by mistake; hence, there has not occurred any incident beforehand.

A major problem arising was the limited ability of the detectors to process the huge amount of data in the ever-growing network infrastructures. (Snapp et al. (1991a))

Although with some systems, a number of the responsibilities are integrated within single components and also some boundaries between them have become weaker, as for example some sensors are performing some pre-processing of the collected data. Already, this separating has basically remained over the decades of Intrusion Detection technology development and evolution.

2.4.1. Initial projects

Distributed Intrusion Detection can be traced back to the early nineties. Heberlein suggested with his Network Security Monitor (NSM) an approach for observing a network segment. It was said to be applicable for broadcasting local area networks and is based on the definition of profiles of usage of network resources and the comparison with the current usage (Heberlein et al. (1990); Snapp et al. (1991a)). One year later, the same research group extended this approach by some distribution features and distinguished between the three components central manager, host manager and LAN manager (Brentano et al. (1991); Snapp et al. (1991b)). The expression central manager already betrayed it is employing a centralised approach and therefore not really applicable for wide area networks since all the high level events are reported to this single manager.

The early distributed Intrusion Detection Systems had one problem in common, namely the difference between network topology and IDS topology. Hierarchies are not very likely to exist in many networks (Balasubramaniyan et al. (1998); White et al. (1996)). Hence, an approach in addition to the existing one with the centralised topology was introduced and developments such as Cooperating Security Managers (White et al. (1996)) with their individual managers were brought into existence, that forces them to coordinate intrusion detection activities themselves. The replacement of the centralised instance is a significant improvement towards the employment of intrusion detection technologies on large-scale networks.

2.4.2. Large-Scale approaches

The first occurrence of the expression *peer-based* in the context of Distributed Intrusion Detection Systems was recognised in the "Cooperating Security Managers" (White et al. (1996)). Although this system shares the basic idea of totally equal managers with our project, it is following a completely different way of distributed intrusion detection. Basically, it was looking to improve the architecture named "Distributed Intrusion Detection System" (Snapp et al. (1991b)), which is tracing users through a specified network domain using

a unique user identifier (namely the Network-user identification) for all sessions a user is maintaining all over this domain (Brentano et al. (1991)). By examining and correlating both, local logging data, such as login and logout information, and network traffic information, such as telnet or rlogin connections, it claims to be able to track users on several machines even when using different user names. The major improvement of the "Cooperating Security Managers" is the substitution of the identifier by maintaining a tail of visited hops for each user. This functionality is said to enable administrators to trace the route of a user through the entire observed network domain.

Another direction of extension was driven by Heberlein with his Internet Security Monitors (ISM) (Heberlein et al. (1992)), which are also based on the Distributed Intrusion Detection System (Snapp et al. (1991b)). It is furthermore utilising outcomes from an earlier of Heberlein's projects, namely the Network Security Monitors (NSM) (Heberlein et al. (1991)). These were initially designed to detect intrusive activity across a local-area network (LAN). By the introduction of a technique called thumb printing with its "extended connections" it is looking to track users through wide area networks. The entire network is divided into several domains and by introducing a hierarchical configuration with its sub domains, users are said to be tracked over an entire wide area network such as the Internet.

2.4.3. Agents, graphs and neural networks

After these initial steps in the early nineties, research on distributed intrusion detection was pushed further and some approaches were brought into being in the mid and late nineties which are more or less all taking advantage of the outcomes of the aforementioned developments. In 1994, Crosbie et al published an approach whose basic improvement was said to be the dividing of the well known monolithic IDS architecture into light weight components; in particular a net of independent agents which may be added or removed to the overall system dynamically. Agents can employ any of the three detection mechanisms: pattern matching; rules based detection and use of genetic programming and are distributed both on a local system and over the local network. Any of the agents may assign certain suspicious levels for special actions and broadcast this to other agents. Basically, no single agent is able to raise an alert, only in correlation with other agents and suspicious behaviour on their side, the threshold can be achieved and alerts will be triggered. (Crosbie and Spafford (1994, 1995))

In 1996 another approach for distributed Intrusion Detection was introduced by Staniford-Chen et al, namely "GrIDS - A Graph based Intrusion Detection System for Large Networks". (Cheung et al. (1999); Staniford-Chen et al. (1996)) GrIDS uses a hierarchical reduction scheme for the graph construction (Staniford-Chen et al. (1996)); and this way the authors

claim the applicability for networks with up to several thousand hosts. The network topology is divided into several domains; each host within them is represented by a node of the graph. In the higher level graph this domain will be represented by a single node on itself. Edges in the graphs are giving information about traffic between the nodes. By definition of suspicious traffic profiles beforehand and capturing the current network traffic in conjunction with matching between both of them malicious behaviour such as worms is said to be detectable.

In the late nineties two new approaches for distributed Intrusion Detection were introduced. Both of them make use of the expression agent for suggesting the independence and autonomy of these modules. Balasubramaniyan assembles his autonomous agents in a hierarchical system in order to generalise and pre-process the data in each of the layers; thus reducing the amount of data reported to the parent node (AAFID) (Balasubramaniyan et al. (1998)). An AAFID system consists of several entities; each of them is type of either agent or transceiver or monitor. Every host may contain several agents but only one transceiver, which controls the agents and to whom all findings are reported. Data is pre-processed and passed to one or several monitors. Monitors may be organised in a hierarchical fashion such that a monitor may in turn report to a higher-level monitor. (Balasubramaniyan et al. (1998)) However, due to the control role of the monitors these single points of failure are a significant drawback. Barrus et al. are also making use of autonomous agents, which are deployed inside one network domain and which are all basically reporting to the same data collector. The novel idea behind this approach is the employment of a hierarchical scheme of escalating levels of alertness. As several times before the users are tracked through the network using a common network identifier and the two values: Danger and transferability, which are thought to be essential in this approach for the alert levels. They are evaluated and challenged against a threshold using a Neural Network. (Barrus and Rowe (1998))

2.4.4. Current approaches

Further research and studies (Vandoorselaere et al. (2004)) in this area are drawing the attention to the problem of leaking of dynamics with traditional Intrusion Detection. Organisational structures in organisations are specific and individual and should be able to be addressed by the topology of the information security infrastructure. With the introduction of so called Enterprise Intrusion Detection Systems (EIDS), a recent development in Distributed Intrusion Detection, these issues could be addressed, and, for example, hierarchical configurations of the processing components became possible. (Danyliw et al. (2003); Fyodor (2000); Vandoorselaere et al. (2004))

Figure 2.1 compares the traditional approach with the one of Enterprise Intrusion Detection

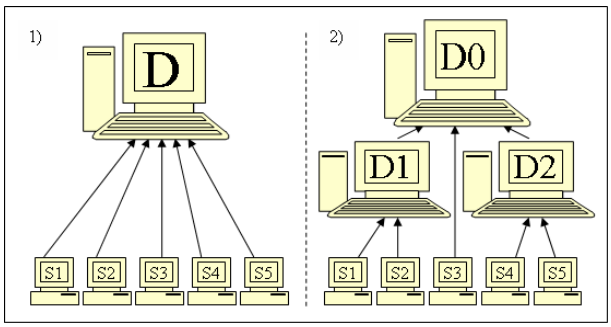


Figure 2.1.: Evolution of Intrusion Detection Technology

and outlines the differences in configuration of the components and the flow of information.

SnortNet

SnortNet (Fyodor (2000)) is an extension to the well-known open-source network intrusion detection system *Snort*, whose development was started in the late 90s. Snort comes with a number of facilities to integrate modules in the form of preprocessors and output plugins. The latter has also been used by the SnortNet extension in order to add distribution features to the approach.

SnortNet was started only 2 years after the initial attempts of Snort itself and has since been discontinued. It was an attempt to add distribution features to Snort by introducing an architecture comprising of network sensors, proxy daemons and a monitoring console. Consequently, hierarchies were the only deployment scenarios supported by this approach. Much attention was drawn to the strict employment of standards at that time, so that protocols such as the Intrusion Alert Protocol (IAP) and the Intrusion Detection Message Exchange Format (IDMEF) have been incorporated (see section 2.6.2 for more details on those formats).

Prelude

The Prelude (ThePreludeTeam (2006); Vandoorselaere et al. (2004)) project targets a different aim with their intrusion detection system by providing a framework for intrusion detection system communication instead of an isolated solution. As it comes with facilities for host as well as network based intrusion detection it can be considered as a hybrid intrusion detection system. Interfaces and libraries are available in order to integrate intrusion detection systems from different vendors and support correlation of event information.

Prelude is an open-source project and lately has gained loads of popularity. An implemen-

tation is available in a stable release and it has been increasingly deployed these days at a steady rate. Besides the components for communication and the required libraries for third party application integration it comes with a web based frontend as a management console. It employs public key authentication and encryption for securing the communication between components.

Feasibility

A number of distributed approaches for intrusion detection systems have been brought into existence, employing very different technologies to support this feature. They are often created for a certain deployment domain only and it is hard to compare them against each other in a straight forward way.

The Columbia University DNAD Team became aware of this fact and has recently carried out research on the feasibility of state-of-the-art distributed intrusion detection systems. (Dnad (2004)) The methodology, they employ, does not simply compare approaches against each other, but moreover evaluates and presents, which methodologies and mechanisms are most appropriate for distributing intrusion detection alert data. The following list of results only provides a brief overview of the outcome of their research:

- A distributed system for intrusion detection system with event data correlation is achievable in general.
- Such a system must not be centralised.
- Bloom filters can be a suitable candidate for preserving privacy of sensitive alert information.
- Uncomplicated and straight forward aggregation and data reduction provides an appropriate level of feedback.
- Employment of network maps seems to be valuable.
- IDS audit data should not only be captured in a distributed manner but also processed and distributed.
- Correlation functions should be provided.
- Employment of text or XML based message formats (such as IDMEF) cause a major overhead on the network traffic.

2.4.5. Distributed Intrusion Detection and Grid Systems

Starting in 2002 a new research area has been discovered by applying intrusion detection system technology to grid systems. Tolba et. al. have published a number of papers to date (Tolba et al. (2005b), Tolba et al. (2005a), Tolba et al. (2002)), describing an idea called *GIDA - Grid Intrusion Detection System*, which is looking to protect existing grid system by implementing IDS technology within them.

The architecture comprises of intrusion detection agents (the gathering components) and intrusion detection servers, which analyse the data and communicate with other intrusion detection servers in order to detect intrusions. The introduction of administrative domains supports supervision of deployment of the agents and their registration with servers. It has been built on top of the Globus Grid Security Infrastructure (GSI) (Welch et al. (2003); Foster et al. (1998)) and is therefore incorporating state of the art standards in the grid context.

2.5. Secure the information channel

Nowadays, solutions for encryption and authorisation are requested more than ever before. With the need for secured communication encryption became more and more popular over the last years. However, basics in this area were already discussed decades ago (Goldwasser (1997); Diffie and Hellman (1976)) and both symmetric as well as asymmetric encryption were worked out. Symmetric encryption algorithms are using the same key for both encryption and decryption, whereby asymmetric algorithms are using a pair of distinguish keys for those tasks, respectively.

2.5.1. Public Key Infrastructure (PKI)

Public Key Infrastructures (PKI) were brought into being addressing and integrating several issues for secure communications. The most important ones are:

- Authentication between the communication participants
- Encryption of all messages
- Signatures for providing non-repudiation and message validation.

Public key infrastructures are utilising asymmetric keys for satisfying their needs in authentication and encryption. According to (Rhee (2003)) nowadays, there are only a very few asymmetric algorithms being both secure and practical in view of their encryption features. Representatives are RSA, ElGamal's Public-key Cryptosystem and Schnorr's Public-key Cryptosystem. Most of the asymmetric algorithms are using primes, logarithms or elliptic

curves due to the need of exhausting calculation power for performing reversal calculations of those operations.

Symmetric encryption could be applied for protection of communications with a large amount of data to be transported. After establishment of the connection with the help of asymmetric keys, symmetric keys can be processed and exchanged over the secure channel. Employment of symmetric encryption yields to less calculation power assumption whereby the security is kept on the same level in comparison to asymmetric encryption. A variety of different algorithms (such as Data Encryption Standard (DES) International Data Encryption Algorithm (IDEA), RC5 or RC6) (Rhee (2003)) may be considered for integration with knowledge grid architectures.

Most representatives of asymmetric encryption algorithms provide another feature, namely the signing of messages. Singing of messages is useful in two very important cases:

- Message Integrity: The message shall be protected against modifications
- Sender non-repudiation: It can be proved with the use of the sender's public key that he / she is the sender of the message

Pre-condition for correct implementation of signatures (and for the public key infrastructure in general) is the strict secrecy of the private key as well as the preservation of the integrity of public keys.

2.5.2. Transport Layer Security (TLS)

Transport Layer Security (TLS) is a protocol, accepted as a standard in form of RFC 4346 (version 1.1 or the former version 1.0 as RFC 2246), for providing privacy and data integrity between two communicating parties. (Dierks and Rescorla (2006); Dierks and Allen (1999)) The Transport Layer Security protocol has been based on the Secure Socket Layer (SSL) protocol (version 3.0), which was developed and published by Netscape.

The specification for TLS version 1.1 (Dierks and Rescorla (2006)) comes with a list of goals it is trying to address, which are:

- Cryptographic security for establishing secure connections
- Interoperability for supporting use of TLS in different, independently from each other developed programs
- Extensibility by providing the framework, in which protocols and algorithms may be easily integrated

- Relative efficiency in order to overcome the problems of cryptographic calculations being very CPU instense

TLS is a layered approach, distributing its functionality over the following two layers:

- The TLS Record Protocol is the low-level layer, sitting directly on top of some reliable transport protocol such as Transmission Control Protocol (TCP). It provides security by employing the properties privacy and reliability for the connection. Technologies such as symmetric cryptography (DES, RC4, etc.) and secure hash functions (SHA, MD5, etc.) are employed for achieving these requirements.

- The TLS Handshake Protocol in turn is making use of the encapsulation of low-level protocols provided by the TLS Record Protocol and supports authentication between communication parties as well as negotiations about encryption algorithms and cryptographic keys. In the end, the TLS Handshake Protocol provides an interface for application layer protocols, which are located on top of TLS, to make use of the secured communication channel and transfer data through it.

2.5.3. Access Control - Protection of information

Protecting information and controlling access for it has long been an issue before information technology has evolved and there was the requirement for incorporating these methodologies with it. Organisations such as militaries as well as businesses have put loads of effort in order to come up with models, which can protect their assets appropriately.

Information technology research has not come up with major inventions within this area; it has rather taken information available for protecting knowledge and applied it to its new environment. Two very popular representatives within this context are the Chinese Wall Security Policy and the Bell-LaPadula model. Each of them will be introduced briefly in its section over the upcoming pages.

Chinese Wall Security Policy

The Chinese Wall Security Policy (Pfleeger and Pfleeger (2003); Brewer and Nash (1989)) has been exhaustively deployed in commercial environments and its utilisation has been set as a legal requirement for certain environments. The major attribute of the Chinese Wall Security Policy is its dynamic approach for protecting information. In practise does this mean that in the first place a party is allowed to access any item of information. The success of access to further information is then based on relations between the information chunks in question.

Technically, the Chinese Wall Security Policy achieves this behaviour by implementing a hierarchical structure for information, using the following terms:

- In the lowest layer of the hierarchy are atomic chunks of information, the so-called individual objects.
- The layer above is made up by grouping into organisations, which results into assignment of information chunks (from the lower layer) to exactly one organisation each.
- In the top-layer of the hierarchy there are the Conflict of Interest classes, which organisations are assigned to.

Basically, whenever one item from within a particular conflict of interest class has been accessed there must not be accessed any other item from within this class unless it is assigned to the same organisation.

The dynamic approach of this security policy makes it "*a subtle combination of free choice and mandatory control*" (Brewer and Nash (1989)).

Bell-LaPadula model

The Bell-LaPadula model (McLean (1985); Wikipedia (2006)) has been introduced as a security model in the early 70s by David Bell and Len LaPadula in order to formalise the US Department of Defence multi-level security policy. It focuses on the confidentiality of the existent classified information.

The Bell-LaPadula model is based on a state machine, which works on the following three entities:

- Subjects - the active part in the model
- Objects - the passive part in the model
- Operations - the actions, which are carried out by subjects on objects

The model is keeping itself in a secure state by allowing secure operations within the system only. This is achieved by employing the following three security properties (Wikipedia (2006)):

- The Simple Security Property making sure that a subject at a certain level of confidentiality cannot read any object at a confidentiality level above its own (no read-up).

- The Star (*) Security Property stating that a subject at a certain level of confidentiality must not write to any object with a lower level of confidentiality (no write-down).
- The Discretionary Security Property uses an access matrix in order to specify discretionary access control.

Although this model has gained loads of popularity since its introduction and is one of the most important security models it is lacking applicability for modern computer systems due to its nature and its restriction to multi-level security policies exclusively.

2.6. Define the way of speaking - Message Exchange Formats

The upcoming approach for this project is making use of communication facilities from various areas, for which the state of the art had to be researched. In detail, the categories for communication can be mirrored by the following list:

- Structured data formats in General - XML
- Data formats in the computer incident context
- Connection establishment

Each of them is covered over the following pages with its own section and, this way, introduces up-to-date developments within its category.

2.6.1. Extensible Markup Language (XML) - Structure data

The Extensible Markup Language (XML) has been introduced in the late nineties and is to be seen as a syntax for creating new markup languages rather than a markup language itself. (Ray (2001)) Over the past decade it has ever grown in popularity, many syntaxes have been developed and it has been utilised in a wide range of application scenarios. This exceptional success of the XML technology can be traced back to the following short list of its features (Ray (2001)):

- *XML can store and organise just about any kind of information in a form that is tailored to your needs.*
- *As an open standard, XML is not tied to the fortunes of any single company, nor married to any particular software.*

- *With Unicode as its standard character set, XML supports a staggering number of writing systems (scripts) and symbols, from Scandinavian runic characters to Chinese Han ideographs.*
- *XML offers many ways to check the quality of a document, with rules for syntax, internal link checking, comparison to document models, and datatyping.*
- *With its clear, simple syntax and unambiguous structure, XML is easy to read and parse by humans and programs alike.*
- *XML is easily combined with stylesheets to create formatted documents in any style you want. The purity of the information structure does not get in the way of format conversions.*

The XML approach defines a way, how information has to be structured. For this purpose there are certain entities existent within the XML language. These are:

- Elements, which are building the blocks of an XML document
- Attributes, which describe elements in more detail
- Namespaces for grouping entities together, for expanding the vocabulary and reusing defined entities.

Many related projects have come into being around the XML technology in order to support operations like creating, processing, transforming or searching within XML documents. The most popular projects within this area are:

DTD – The Document Type Definition, which defines the names of elements with their attributes and specifies rules for their combination and sequence.

XML Schema's – An alternative to DTD for describing the structure of an XML document; but in contrast to a DTD it is encoded in XML itself.

Sax – The Simple API for XML for parsing documents and processing its information by defining event handlers for certain data chunks.

DOM – The Document Object Model for parsing an entire XML document and loading its content in form of a tree structure into the memory.

XQuery – For searching an XML document for pre-defined text patterns.

XPath – For accessing information within an XML document by defining the exact path towards the desired item within the corresponding tree.

XSLT – XSL Transformations for transforming data within XML documents into a new representation such as another XML document, a HTML document, programming language source code, PDF documents, etc..

DOM

As previously stated, the Document Object Model is a way to keep the content of an XML document in the memory in form of a tree structure. Nodes can be accessed easily by browsing through the tree using child and parent relationships for entities. Transformation between DOM and XML may be performed in both directions; consequently, DOM can be used to parse and process an existent XML document as well as to create a new tree structure and output a new XML document from it.

DOM was originally released by the W3C as a recommendation for a standard tree-based programming API for XML documents. After starting up with implementations for Java and JavaScript it has become more and more popular and may now be seen as a general purpose XML API for many applications such as editors or file management systems. (Ray (2001))

Unlike SAX, which utilises call-back technology for events, DOM provides facilities to create and modify objects within the tree. DOM modules are available for mapping the corresponding XML entities. It has been well-defined which modules are allowed to maintain what kind of relations with which other modules.

2.6.2. Exchange formats for computer incidents

In order to allow wide implementation of the approach, the protocols and interfaces between the components have to be well-defined. There is currently a lot of research in both the area of exchanging information in a secure manner and in defining message formats for security related messages. (Bellovin et al. (2003)) Lots of standards and protocols have been developed such as Intrusion Alert Protocol (IAP) Gupta et al. (2001), Incident Object Description and Exchange Format (IODEF) (Demchenko (2003); Danyliw et al. (2006)), the Intrusion Detection Exchange Protocol (IDXP) (Feinstein et al. (2002)), the Intrusion Detection Message Exchange Format (IDMEF) (Curry et al. (2002); Debar et al. (2005, 2006)) and the Blocks Extensible Exchange Protocol (BEEP) (Rose (2001)). Each of them will be addressed over the following sections briefly.

Blocks Extensible Exchange Protocol (BEEP)

The Block Extensible Exchange Protocol (BEEP) is a "*generic application protocol for connection-oriented, asynchronous interactions*" and "*permits simultaneous and independent exchanges within the context of a single application user-identity*". (Rose (2001)) The messages carried within BEEP are arbitrary MIME content (Multipurpose Internet Mail Extensions) and usually XML encoded. Security for BEEP is provided using the Transport Layer Security (TLS) protocol.

BEEP can be mapped into different underlaying carrier protocols such as Transport Control Protocol (TCP) connections. The Blocks Extensible Exchange Protocol uses so-called profiles in order to define a connection between two parties; these connections are called channels. Two different kinds of channels are existent, namely the initial tuning channels for setting up the connection and continuous channels for exchanging data.

BEEP connections are stateful connections and can either be one-to-one or one-to-many exchanges. One-to-one exchanges are used if the reacting peer only needs to send a single reply to the initiating peer or answers a request with an error message. One-to-many exchanges are established whenever a reacting peer keeps on sending answers to the initiating peer whilst performing the requested task. In the end a termination message from the reacting peer indicates the end of the reply.

Intrusion Alert Protocol (IAP)

Regarding (Gupta et al. (2001)) the Intrusion Alert Protocol "*is an application–level protocol for exchanging intrusion alert data between intrusion detection elements, notably sensor/-analysers and managers, across IP networks*". It is making use of the Intrusion Detection Message Exchange Format (IDMEF) (see following sections for details) as format for the transmitted alerts. Noted from recent developments for the RFCs, IAP has not become a standard in favour of the superior Intrusion Detection Exchange Protocol (IDXP) (see following sections for details), which has been developed for the same purpose.

The Intrusion Detection Alert Protocol has been designed with transport and security issues in mind in order to provide facilities for sending sensitive alert data across IP networks. Additionally, it comes with options for future extensions in order to support developments for sensor / analyser configurations or response transmissions. In order to provide reliable and sequenced delivery of data between nodes IAP uses the Transmission Control Protocol (TCP) as underlaying transport layer mechanism.

The communication may either be established between the sensor / analyser and manager directly or by passing traffic through proxies and gateways. Furthermore, it makes use of

Multipurpose Internet Mail Extensions (MIME) (which are well-known from current emailing systems) in order to denote the type of alert data. There are always two TCP connections established; one for each direction of traffic to pass using the protocol. Security is provided using the Transport Layer Security (TLS) protocol (see section 2.5 for details).

Intrusion Detection Exchange Protocol (IDXP)

The Intrusion Detection Exchange Format Working Group (IDWG) has developed the Intrusion Detection Exchange Protocol (IDXP), "*an application-level protocol for exchanging data between intrusion detection entities*". (Feinstein et al. (2002)) It provides facilities for mutual-authentication, integrity and confidentially over a connection-oriented protocol. Although it is prepared to transport all kind of data such as unstructured or binary data IDXP is supposed to carry messages in Intrusion Detection Message Exchange Format (IDMEF) (see following section).

The Intrusion Detection Exchange Protocol is directly linked to the Blocks Extensible Exchange Protocol (BEEP) protocol in form of specification of a profile for it. Consequently, certain issues such as confidentiality and connection control, which are addressed by the BEEP already, do not need to be implemented for IDXP again, as it can benefit from them due to its utilisation of BEEP. Security for IDXP is provided in correlation with BEEP by using protocols such as Simple Authentication and Security Layer (SASL), Tunnel profile and the Transport Layer Security (TLS) protocol.

Communication between two IXDP parties are only existent in pairs and are always established using the BEEP protocol, by passing information through one ore more BEEP data (continuous) channels. The peers involved in the communication may only be managers or analysers. Due to the employment of the BEEP, peers may either communicate directly with each other or indirectly through one or several proxies.

The Intrusion Detection Exchange Format (IDMEF)

The Intrusion Detection Exchange Format Working Group (IDWG) released an internet-draft for the Intrusion Detection Message Exchange Format (IDMEF) for the purpose of *defining data formats and exchange procedures for sharing information of interest to intrusion detection and response systems, and to the management systems which may need to interact with them.* (Debar et al. (2006)) In contrast to the previously discussed protocols, which are addressing the connectivity of the communication, IDMEF is dealing with the content of the messages and describes the structure of the transmitted data.

The IDMEF protocol was designed in order to establish a standard of data representa-

tion for the communication channel between intrusion detection analysers (or sensors) and managers (or consoles) for the purpose of improving interoperability between products from different vendors. The Intrusion Detection Exchange Format has to be understood as an object-oriented representation of alert data exchanged between intrusion detection components.

An IDMEF message may either be an *Alert* or a *Heartbeat*. In contrast to the rather simple type heartbeat the Alert class is a complex type and maintains relations to many other types in order to include information about analysers, sources and targets with their details, timestamps, classifications as well as assessment for the event. IDMEF in its current status defines the structure of each of these entities by providing detailed Document Type Definitions (DTD) for them.

Incident Object Description and Exchange Format (IODEF)

The IETF Extended Incident Handling Working Group (inch) has released in turn an internet draft for describing computer incident data, which, in contrast to IDMEF however, describes data on a far higher and more abstract level. The intention by the group is to "*define a data representation that provides a framework for sharing information commonly exchanged by Computer Security Incident Response Teams (CSIRTs) about computer security incidents*". (Danyliw et al. (2006))

As one can see the deployment environment for IODEF is communication channels between CSIRTs rather than data transmission between components of single side intrusion detection system deployments. Consequently, it focuses on representation of abstact incident data rather than detailed IDS raw event information. As this incident information may be very different depending on the deployment environment, IODEF has been created as a framework to convey commonly exchanged incident information, which may be adopted for specific needs.

As all the other mentioned technologies the Incident Object Description and Exchange Format is an XML encoded format, for which an XML schema has been defined. All information for this data format is carried inside an IODEF document and incidents are represented by their own classes with a number of relations, one instance of the incident class for each of them. The information attached to an incident in form of relations is made up by items such as timestamps, descriptions, assessments, contact information, history and event data. Usually, one incident instance may maintain relations with many of those entities.

2.6.3. Establish connection

The last part of the communication category is made up by the transmission protocols themselves. As this is a very wide and exhaustive area, a full overview of all available technologies cannot be provided. Instead, I focused on the approaches, which were believed to be of any use for the project.

The following list of approaches has been researched further:

- TCP/IP sockets for low-level communication establishment
- Remote Procedure Calls (RPC) in General
- SOAP as one representative of RPC for high-level communication establishment

Each of the technologies will be presented with their sources in more detail in the upcoming three sections.

TCP / IP Sockets - Transfer Control Protocol / Internet Protocol

TCP / IP - Sockets (Stevens (1994)) are connection-based communication channels, which are established on ISO OSI layer 4 - Transport Layer.

Firstly, the server side of the communication has to bind a TCP socket on its local side and wait for incoming connections. Afterwards, the client side of the communication uses the IP address of the server and the just before assigned TCP port to connect against this so-called socket. At the same time the operating system of the client will dynamically assign a port for the client so that the entire communication uses two couples of IP address and port number for addressing.

The operating system is in charge to pass on the incoming messages to the corresponding application regarding the port number applied. TCP-IP is a connection based protocol; meaning that the protocol implementation itself takes care that information arrives in order and is resubmitted in case of any problems. The Transfer Control Protocol is by far the mostly utilised transport layer protocol in the Internet.

Remote Procedure Calls (RPC)

Remote Procedure Calls (RPCs) are a well-known technology for software development of distributed infrastructures in order to access intelligence available on a remote location. Initially intended to be used for invocation of functions available on a remote node, they are often used for exchange of information due to their ease of use.

Remote procedure calls have been implemented in many different protocols, such as:

- Common Object Request Broker Architecture (Corba) - Very mature and well-known cross-platform and language-independent RPC approach
- Remote Message Invocation (RMI) - The java implementation for RPCs
- Simple Object Access Protocol (SOAP) - An XML based, cross-platform and language-independent RPC mechanism

As XML has been identified as a core technology to be used for this project only the Simple Object Access Protocol (SOAP) is discussed further in its own section.

SOAP

The Simple Object Access Protocol (SOAP) is mainly used for getting the data from one place to another within web-services infrastructures (Newcomer (2002)) and has gained loads of popularity because of this technology over the last years. It comes as a combination of the two technologies web with Hyper Text Transfer Protocol (HTTP) and the Extensible Markup Language (XML).

In a more technical sense, SOAP may be understood as an extension to the Hyper Text Transfer Protocol (HTTP) in order to support XML messaging. In contrast to HTTP, which sends a request in order to get a HTML document in return for displaying it in a web browser, SOAP sends an XML formatted request and receives an XML encoded reply from the server via HTTP response if successful. In the end, the existent web server must be extended by a SOAP module in order to understand and process incoming SOAP messages.

Actually, the specification of SOAP allows transmission of the messages using other carrier protocols although HTTP is the only defined one. A SOAP message is made up by a SOAP envelope which, in turn, consists of an optional SOAP header and a compulsory SOAP body. The header is in place for carrying the attributes for a message such as quality of service. The body, in contrast, contains one or more body blocks comprising of the message itself.

2.7. Grid Technology

The definition for grid has evolved over the last few decades. Len Kleinrock, for example, predicted grid technologies as recently as the late sixties with the following statement: "*We will probably see the spread of* computer utilities*, which, like present electric and telephone utilities, will service individual homes and offices across the country*" (GRIDSTART (2003)). Later on, grids were described the following way: grid is "*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations*" (Foster et al. (2001);

Foster (2003)). A very recent definition explains grid technology as "*supporting the sharing and coordinated use of diverse resources in dynamic VOs – that is, the creation, from geographically and organisationally distributed components, of virtual computing systems that are sufficiently integrated to deliver desired QoS*". (Foster et al. (2002, 2001))

2.7.1. History and evolution

When starting the research into grid technology about a decade ago most of the work was mainly focused on the so-called low-level or connectivity projects. As time passed more and more projects have turned towards development of applications on top of these fabrics. According to (Bubak et al. (2003); Graham et al. (2004)) several waves may be defined, which mirror this evolution of grid technology. Finally, the community came up with a de facto standard called Globus and a corresponding implementation called Globus Toolkit (GRIDSTART (2002)). Taking Globus and its implementation as a base and extending and modernising it have led to a new architecture called Open Grid Services Architecture (OGSA) (Foster et al. (2002)).

Most modern grid networks are intended to either store and share huge amounts of data or perform the calculation of certain problems distributed on several locations inside communities or so-called virtual organisations. The sharing of knowledge among participants is not yet addressed efficiently, although there is obviously a wide range of possible deployment scenarios and applications that can be considered:

- A grid application for interconnecting different Intrusion Detection Systems at very different geographical locations all over Europe to establish a European Defence Community which looks to face modern threats as a unit rather than the currently established approach of island solution, where each organisation (no matter which type of, university, commercials or governmental institutions) try to create and maintain their own pool of information. When sharing one organisation's gathered information throughout a European wide network it is believed that this will contribute to the security situation, significantly. This approach is discussed in more detail by (Pilgermann and Blyth (2004, ISBN: 0-9547096-2-4); Vidalis et al. (2003)) and shows the need for secure and reliable communication for a knowledge based grid environment.

- A grid application for sharing health information between several health related organisations, for example, inside the European Union for accelerating the process of requesting very important information for a specific topic and to enable individuals to publish their knowledge and outcomes into a network and contribute this way to the

community (Vidalis et al. (2003)). A related approach is proposed with the EU project *European Federated Mammogram Database Implemented on a grid Structure* (Bubak et al. (2003)), although this one is still based on a traditional grid topology.

- While investigating current projects inside the European Union more and more projects seem to turn from the need of a traditional computational grid environment towards an approach for sharing information. Another candidate; from our point of view, is the EU funded project Astrophysical Virtual Observatory (AVO) which seems to request similar functionality. (Bubak et al. (2003))

- Wide employment may be considered in almost all areas of science, commerce and even lifetime since all these environments handle certain amounts of data. For example, projects in genetics and proteomics, multimedia data archiving and financial modelling are thought to be implemented using knowledge-based grid technology (Talia (2004)). Whenever the publication, merging or integration of knowledge is thought to be reasonable, a knowledge based grid approach might be considered to share the information efficiently.

It was already discovered in 2002 that there is a *shift from computationally intensive applications [...] to knowledge discovery and categorisation.* (Graham et al. (2004)) However, from our point of view, no modern grid middleware is able to address the issues arising for this kind of problems properly. Cannataro et al discovered the lack of addressing the issues of sharing knowledge in grids and, consequently, introduced their *Knowledge Grid* (Talia (2004); Cannataro and Talia (2003)), which is, basically, more or less a toolkit for developing knowledge grid applications on top of available grid architectures and does not address the drawbacks of modern grid environments for knowledge sharing itself.

Grid has become a catch word over recent years and a variety of projects are decorated with various names for this kind of technology. Foster developed a three point check list (Foster (2003)) for the use of the technology grid for approaches, and the mirroring against them justifies the right location of this project in the grid environment:

- No centralised control: a key issue for our knowledge based grid is the total avoidance of central instance for any purpose.

- Use of standard, open, general-purpose protocols and interfaces: Basically, the same protocols and standards are utilised as they are employed in the OGSA project. Finally, the whole project will be open and based on standards without any exceptions.

- Delivers non-trivial qualities of service, which means *the utility of the combined system is significantly greater than that of the sum of its parts* (Foster (2003)): It is very reasonable that the intelligent and adequate integration of knowledge from different locations will build up a large knowledge base, which may, finally, be much more beneficial then the isolated information of each node. In conjunction with decent data mining and data merging facilities, big enhancements can be achieved.

2.7.2. Integration with Web-Services

In the beginning of this decade, people became aware of the large number of similarities in terms of aims and methodologies for the two technologies GRID and web services. Consequently, it was decided to integrate the two technologies with each other and continue future developments together. Previous outcomes from the research and development within the grid community such as the Grid Services Infrastructure (GSI) were reused and in 2004 a draft for the new specifications for the so-called Web-Services Resource Framework (WSRF) has been submitted to the *Organisation for the Advancement of Structured Information Standards* (OASIS) and was finally approved as a standard in 2006 in version WSRF v1.2. (Czajkowski et al. (2004b))

The WSRF comes, as its name says, as a framework made up by several components, namely:

- The WS-ResourceProperties for describing stateful resources and web services and the way these ones can be accessed.
- The WS-ResourceLifetime in order to support immediate or instant destroying of WS-Resources.
- WS-RenewableReferences for the purpose of annotating a new WS-Addressing endpoint for a resource in case a current endpoint becomes invalid.
- WS-ServiceGroup for grouping web services in the form of heterogeneous by-reference collections.
- WS-BaseFault for reporting errors.
- WS-Notification for standard approaches to notification.

Many of the concepts and technologies that had been used for OGSI such as SOAP as communication protocol or Web Service Description Language (WSDL) (Christensen et al. (2001); Chinnici et al. (2006)) have been reused for WS-RF. In most cases, the features of

OGSI are directly mapped into a location of the Web-Services Resource Framework. A major advantage of the WSRF, however, is its modularity, which overcomes a major drawback of OGSI, namely its complexity.

2.8. Data persistence

Since the early years of information technology, storage of information has always been a major issue; meaning that data and processed results had to be made persistent in order to reuse them at a later time. Early developments for data persistence were based on punchcard technology. Following this, improvements could be made by using magnetic devices, which are now in turn superseded by optical devices.

Detached from the developments in physical devices, many efforts have been put on introducing layers for structuring the data and its representation to the user. This way, files were introduced, which are structured in folders, which all together make up filesystems. Since then, there have been brought into being a variety of file system types for different purposes. The following list is only a selection of very popular representatives:

- VFAT – File system type, which is commonly used for removeable media, such as floppy disks or memory sticks
- FAT16 / FAT32 (File Allocation Table) – commonly used file system type for early windows versions
- NTFS (New Technology File System) – MS propriety file system type, which is utilised in latest versions of the Microsoft Windows operating systems
- Ext2 / Ext3 (Extended) – Very popular and mature file system types for the Linux operating system
- ReiserFS – State of the art file system type for Linux operating systems implementing journal file system technology

Persistence of information inside files is nowadays the first and easiest step to keep data. With the need for storing large amounts of data and provide more sophisticated access facilities database technology has been introduced, which, basically, provides another abstraction layer between the filesystem and the user as it stores its information, which is mainly made available in form of tables, in files as well.

2.8.1. Databases

Database management systems are software that manage databases by storing data and making that data available to several (concurrently requesting) clients for reading and modifying. It comes with a lot of features, which make them a compulsory component in state of the art software projects. A selection of these are:

- Structuring of data and mapping of different data types
- Mirroring relations between data chunks and ensure correctness
- Management of roles and access control for resources
- Consistency as well as savepoints, delayed commit and rollback mechanisms
- Easy access using a standard language called Structured Query Language (SQL)
- Transactions for bundled execution of a set of statements
- Replication and distribution to different geographical locations
- Backup and recovery mechanisms
- Change and Access logging
- Automated optimisation for executing statements

Starting up the developments with relational database systems there are different kinds of database managements system available these days, which are:

- Relational database management systems
- Object oriented database management systems
- XML database management systems

Each of them is introduced briefly in a section over the upcoming pages.

Relational database management systems (RDBMS)

Relational databases, the original form of databases, store tables of data. Regarding to (Quin (2000)) these tables have the following characteristics:

- It consists of many small pieces of information

- Information is kept in atomic pieces within cells
- Every row within the table is considered to be complete; a column, however, spans with its heading over all entries, saying rows, within the table

The major advantage of relational databases has been its option to establish and maintain relationships between the tables, which are also called objects. Two attributes of relationships are mapped into the databases, namely the type and the cardinality of the relationship.

With the introduction of relational databases the corresponding language to access data inside them had been released as well, namely the Structured Query Language (SQL). Its developments can be traced back to the late 70s; however, its current version has been accepted as a standard by the International Organisation for Standardisation (ISO) in 1992. Although, vendors of database management systems apply slight modifications and extension to SQL, it is a common language to access data from relational databases nowadays.

Object oriented database management systems (OODBMS)

With the high popularity of object oriented approaches for modern information systems there was a growing need for making data of these systems persistent in an efficient way. The utilisation of relational databases for storing information from object oriented software architecture, as performed initially, turned out to be insufficient as RDBMS are not prepared by design to hold this kind of information and, as part of the development process, loads of efforts have to be put on the mapping between the software systems internal data model into the relational data model, used by the database, which is based on tables. Object oriented databases have been looking to overcome these problems by storing the data as it is existent in the information system already and, this way, enable the developers to utilise one consistent database model throughout the entire information system. (Quin (2000))

Early research on object oriented databases can be dated back to the late 70s; however, releasing of products within the context had only been started in the late 80s. Since then, there has been a significant growth in both, the number of products offered by the vendors and the requests of developers and users for this technology. A major advantage of OODBMSs is their ability to map very complex relationships between objects; consequently, it has become more and more popular in industrial sectors such as e-commerce, engineering product data management and medicine. The major drawback of OODBMS technology is its incapability to process large amounts of data, which leads to the following conclusion for the presented database management systems so far:

- For high-volume, low-complexity data sets one should use relational database management systems

- For high-complexity, low- or reasonable volume data sets one should employ object oriented database technology

Technically, the developer may benefit from the object-oriented features, supported by OODBMS, such as inheritance and versioning.

XML database management systems

As much as object oriented software design has been established as state of the art for software development, the Extensible Markup Language (XML) has been becoming more and more popular for structuring data for information exchange. XML as the core technology has been addressed within this chapter already (see section 2.6.1 for details). However, with the growing popularity of this technology a requirement has been identified to make XML encoded information persistent on one hand, and transform data inside database into XML encoding when querying the database on the other.

Interfaces have been created, located on top of databases, which are capable of processing requests to the database, which are XML encoded on the one hand, and which return results from the enquiry in XML encoded format in turn. No standard has yet been established for this approach, but a number of companies and organisations have introduced their own concepts employing this idea, such as:

- The Microsoft Software Developer Network has introduced a technology called T-SQL, which uses an XML document for both the querying of a database as well as the returning of the corresponding results. (Xynos (2005); Dobson (2004))

- Sun Mircrosystems makes use of mapping from SQL into an XML structure for reasons of persistence for their enterprise java beans (EJB). (Xynos (2005))

XML encoded representation of database content can benefit significantly towards the processing of information as it contains the structure and meaning of entities itself. Furthermore, it facilitates the process of transformation as an XML document is a very good starting point for transforming into other representations (such as a new XML document, HTML, PDF, etc) by using technologies like XSL Transformations (XSLT) (see section 2.6.1 for more details).

2.9. Conclusion

Within chapter 2 a detailed overview has been presented about history in related research areas as well as state of the art projects and technologies for them.

First of all, a general overview about security in information technology has provided an overview and put the technology intrusion detection into context. An introduction into intrusion detection system technology has presented the concepts, as well as types, available for this technology. The topic has been narrowed down further by focussing on distributed intrusion detection systems and the evolution of these systems has been presented chronologically with an exhaustive list of available projects. The second major topic named grid technology has been introduced by providing some information about the history and presenting the standards within this area. Recent developments towards co-operation with web services technology have been addressed.

Besides the two major topic IDS and Grid there are a number of projects and research areas, which this project needs to gain information and knowledge from; in detail those ones are:

- Network communication and its protocols and standards
- Securing the network connections for ensuring privacy and integrity
- Message formats for sending information in a structured manner
- Data persistence in general for evaluating recent database technologies

With the available information about state of the art projects within (distributed) intrusion detection systems as well as grid technology, the limitations and potential for further developments may be worked out. Chapter 3 presents the objectives for this project in detail and introduces a common set of vocabulary for the project. Afterwards, the chapters lay down, how to design, implement and evaluate the methodology and, finally, chapter 8 gives details about the achieved results.

Chapter 3.

Objectives and Terminology

3.1. Introduction

In the previous chapter the state of the art in the area of distributed intrusion detection has been presented in intricate detail. Within that chapter the lack for exchange security related information such as intrusion detection system audit data across organisational boundaries has become clear already.

With this chapter the idea of inter-organisational intrusion detection will be presented in more detail. First of all, the presentation of three very different deployment scenarios shows the necessity of inter-organisational intrusion detection system technology in modern network environments. Afterwards, the objectives for such an approach are presented; whereby the connection to modern grid systems will be drawn. The chapter is finished off with an overview of expressions and ideas, which are essential for the presented IOIDS approach. They provide a common vocabulary in order to understand the chapters 4, 5 and 6, which contain technical details of the architecture and implementation.

3.2. Deployment Scenarios

Inter Organisational Intrusion Detection is thought to be beneficial in very different employment areas. From the interconnection of event loggers from different companies up to the connection of computers of home-users connected to the Internet is considerable. The three following scenarios have been chosen in order to present the flexibility in employment environments for IOIDS:

1. Interconnection of Intrusion Detection Systems for an entire Supplier-Consumer chain.
2. Integration of security related information from academic institutions on the one hand or collections of commercials on the other from all over the Internet for improving detection rates and counter measuring.

3. Open communities for private end user protection.

The following three subsections describe the mentioned scenarios in detail.

3.2.1. IOIDS in Supplier-Consumer chains

After implementation of Inter Organisational Intrusion Detection in near future, a requirement for providing security related information might be part of agreements between several parties, such as convenient for the relationship between suppliers and consumers, and finally, the entire chain of suppliers, parts of a product travel until it ends up at the final consumer. Figure 3.1 pictures the problem in more detail:

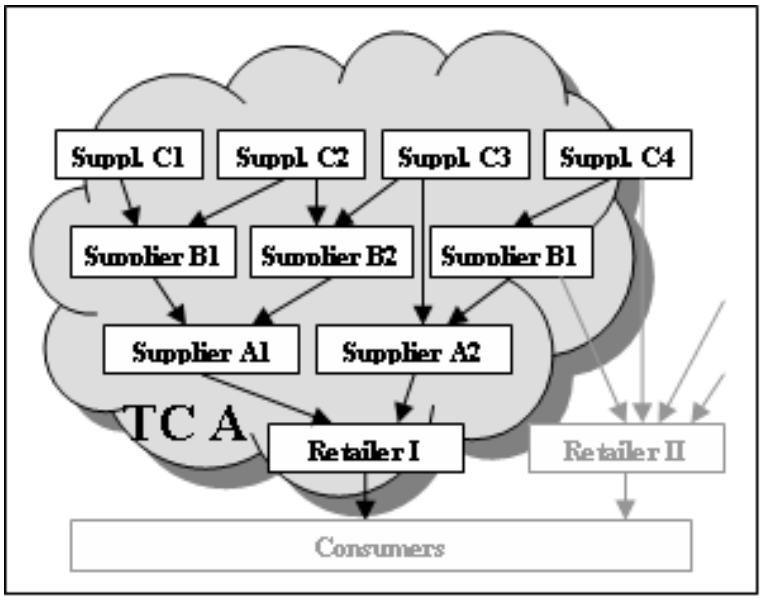


Figure 3.1.: Supply Chain Scenario

In the drawn picture Retailer I might simply demand for each of its potential suppliers to provide a certain amount of security incident related information about the organisations' infrastructure. Moreover, the suppliers of the suppliers are requested to provide the same kind of information. This approach will be pushed to a certain grade and the retailer can finally make sure that it will be informed whenever some kind of significant behaviour has occurred. A policy will be brought into being, which the supplier on the one hand but also the retailer on the other have to align to. Finally, a Trusting Community (in the example named TCA) will be brought into being.

The following requirements for this kind of infrastructure become obvious already when considering this situation only:

- Each node must be totally confident about the location, the data was originated at, and based on this knowledge, decides about the processing and integration of the data.
- Information from several types of audit data generating applications must be able to be processed and integrated.
- Any supplier or retailer must be enabled to share information with nodes from several Trusting Communities.

3.2.2. Sharing of security related information between academic institutions

Another deployment scenario draws the attention to the problem of missing facilities for exchanging security related information between organisations on a very abstract level. Although organisations such as Computer Incident Response Teams (Cert) (Cert/CC (2004)), Common Vulnerabilities Exposes (CVE) (CVE (2004)) and Information Sharing and Analysis Centres (ISACs) (IT-ISAC (2005)) are supporting this process by providing common names and descriptions the actual process of publishing and gathering information is left to humans; hence, it involves a reasonable amount of manual intervention.

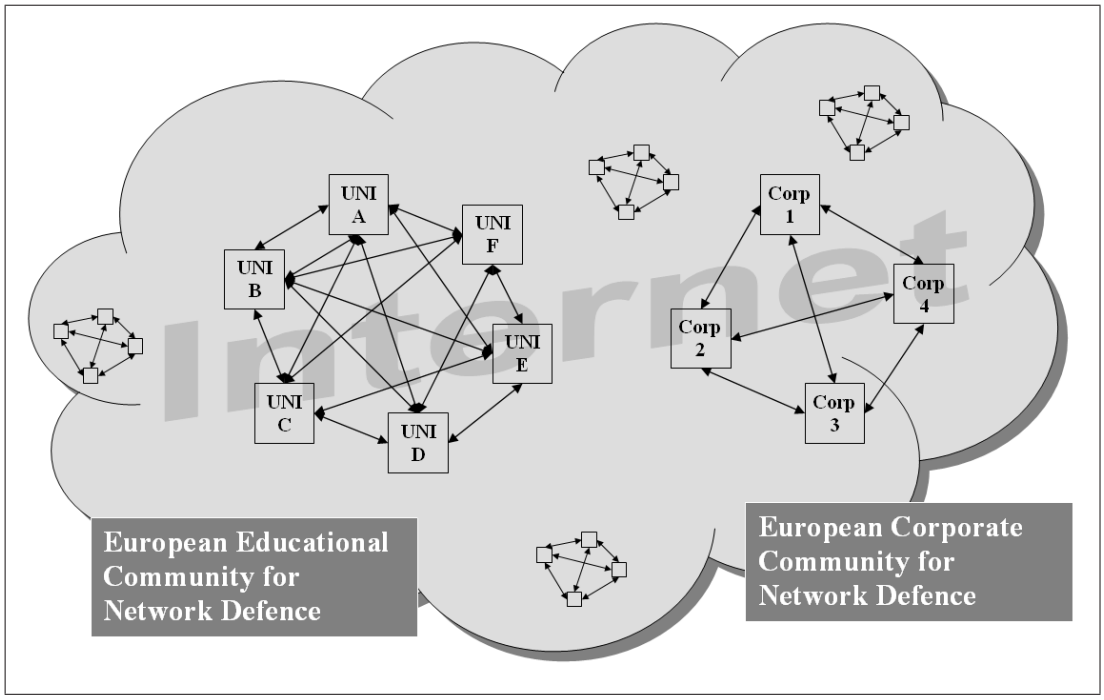


Figure 3.2.: Inter Organisation IDS in communities

Introducing a "European Educational Community for Network Defence" would be able to

cut down this amount of time observably. After agreeing upon a policy for exchanging security related information (including high-level message formats, roles of nodes, trust relationships, etc.) and deploying them into a community specific protocol, knowledge will be exchanged automatically.

The same approach is considerable for a reasonable number of companies throughout a certain geographical area such as the European Union. Once they became aware of the potential benefits of exchanging incident, attack and countermeasure information between each other, they start exchanging this type of knowledge and will face the problems of modern threats as a unit rather than isolated from each other.

The two scenarios do not exclude each other. This way, certain universities might also contribute knowledge to the corporate community or vice verse. After all, it will depend on the policies being in place in both communities. Trust within these so-called Trusting Communities must never be undermined by any node.

In fact, the information or knowledge maintained and provided within as well as across the communities belongs to everybody and nobody. It is comparable with the approach of open source software, which has been gaining lots of popularity over the recent years. Everybody may contribute and benefit from the infrastructure and no single node can take it down. The behaviour of a community is based on the policy, which the initial members have to agree upon.

3.2.3. Open communities for private end user protection

Nowadays, also end users become more and more aware of the threats they are facing when connecting their machines to the Internet. Enlightenment and availability of free and open-source software for measures such as anti-virus and personal firewalls have provided an enhancement of protection for home computers. (NSS (2005)) Ease of use and simple configuration are pushing this process.

However, Intrusion Detection Technology has not yet reached a significant penetration in the end user market. A lightweight approach for an IOIDS architecture is able to change this situation significantly. End users are no longer depending on the decisions of companies, instead they can create their own communities and, this way, exchange security related information among each other. They do not need to trust another party about up-to-date information, but they may decide themselves, which Trusting Community they want to join and which members they want to trust as a reliable source of information.

3.3. Objectives

The subjacent communication infrastructure has many attributes in common with currently available grid systems. However, due to its nature of sharing knowledge instead of calculation power or storage capacity it cannot be considered directly as such one. Consequently, our modified version of a grid infrastructure has been named knowledge grid. Besides the attributes, this modified infrastructure has in common with traditional grid systems, some are novel and unique to the knowledge based approach. The following objectives were grouped for this approach of a knowledge based grid infrastructure:

3.3.1. Knowledge Grid in General

- Information (knowledge) sharing as the major goal on top of all issues arising.
- Since the exchanged knowledge may consist of very confident or sensitive information, security has to be taken into account as an important issue. This includes but is not limited to confidentiality, encryption, authentication, authorisation, message validation and non-repudiation. Research on security for grids has been undertaken by the Open Grid Service Architecture Security Working Group (Nagaratnam et al. (2002)) and especially for communities in "The Anatomy of the Grid - Enabling Scalable Virtual Organisations" (Foster et al. (2001)) and "VOMS, an Authorisation System for Virtual Organisations" (Alfieri et al. (2003)) and their achievements will also be used for addressing the security goals in knowledge based grid environments.
- Reliability is addressed by "ranging from client-server to peer-to-peer technology" (Foster et al. (2001)). Moreover, consistence of the information must be guaranteed throughout the whole grid network.
- It is very important that the members of a community may trust the shared information and that they are furthermore able to assess the trustworthiness of information based on the source of information. This can be achieved using trust relationships which were in a similar way already described in (Foster et al. (2001)) as "flexible sharing relationships".
- Knowledge grids must scale from small locally employable up to internet-wide scenarios.
- Easy deployment for a wide range of applications and the ensuring of interoperability and flexibility. This includes (Foster et al. (2001)):

  - The concealment of functionality for overlaying implementations in all layers of the approach.
  - An approach totally based on standards and the creation of an open architecture for employment with other implementations
  - Easy to use interfaces

3.3.2. Objectives in common with traditional GRID technology

Furthermore, objectives already described in (Foster et al. (2002)) have to be employed, too:

- Error notification and error handling including rollback mechanisms for ensuring consistency of information throughout the entire grid network
- Address naming and address resolution. In this point of view further issues arise when taking into account the requirement of protecting identities as described in more detail in (Pilgermann and Blyth (2004, ISBN: 0-9547096-2-4)).
- Upgradeability and compatibility between versions.
- Authorisation and controlling the flow of data.
- Concurrency control, on the one hand within communities, but also, on the other hand, for traffic across community boundaries.
- Scalability, which enables the employment of the approach for both local and large scale networks.

3.3.3. Objectives not adopted from traditional GRID technology

More objectives are often described for modern grid topologies, which are not considered to be compulsory for a knowledge grid environment due to the differences between traditional and knowledge based grids. These include:

- Quality of Service - since we are not dealing with calculation power or storage sharing, the Quality of Service might only be applied to the whole grid network in the view of availability and the like, but for the knowledge services QoS is not thought to provide an added value.
- Since everybody contributes knowledge to the grid and also gathers information from it no accountability is considered to be employed so far.

- Delegation of authentication credentials are essential in traditional grids for performing processes on behalf of users and, this way, initiate new processes on other locations with these credentials without verifying them each time again. For knowledge based grids, however, this issue is not thought to be of high interest.

3.3.4. Objectives in particular for distributed Intrusion Detection Systems

Besides the issues arising for the subjacent communication platform the ones for the distributed intrusion detection system application running on top of it have to be defined as well. After introducing a list of generic aims for distributed intrusion detection systems four subsections will draw more attention to issues arising for such a wide-scale approach.

In general a distributed intrusion detection system architecture should satisfy the following list of objectives:

- Completeness of detectable attack patterns – in section 2.3.2 several detection technologies for intrusion detection systems have been introduced. It is desirable for an IDS to cover as many of those technologies as possible.

- Correlation of event information – intrusion detection event information is not processed in an isolated manner on one side anymore; instead, event information is correlated over several domains in order to improve detection and minimise false positive rates.

- Stateful inspection – one important feature of network intrusion detection systems is the ability to identify attacks, which span over several network packets (segments, datagrams, fragments, etc).

- Reporting and user interaction – besides the notification of the user using alerts in near-realtime, reporting facilities should be in place and the user should be able to interact with the system in order to apply configurations or initiate other reactions.

- Response and countermeasure – on top of the reporting and alerting facilities IDS are often able to carry out actions in response to certain attacks. These often end up in countermeasure procedures and are not rarely carried out with the help of other nodes within the network (firewalls, routers, operatings systems).

- Integration with existent IT security infrastructure and management environment – security can only be achieved by the combination of a variety of different security measures, which should be able to integrate with each other. In the end it is beneficial to integrate all those in a central management environment.

- Realtime distribution – in order to react on time to malicious behaviour, knowledge must be available as soon as possible - preferably in realtime.
- Dynamics for distribution domains – different kinds of knowledge may require different domains for distribution. Facilities should be in place, which satisfy this need in a simply configurable manner.
- Duplicate data identification and avoidance – especially in the context of domains with the ability for knowledge to travel across domain boundaries, measures have to be in place in order to avoid the processing of the same knowledge several times.

Message types

When dealing with distribution of intrusion detection audit data in the context of knowledge-grid systems the messages exchanged within the systems change from simple representation of raw events (as supported by the well-known standard IDMEF) to more complex and dynamic message formats. The following two message types became necessary for an IOIDS infrastructure:

Knowledge Requests – Due to a certain situation the local node identifies the need for more information. Consequently, it populates a request with certain parameters to gather more knowledge. Other nodes may reply to this messages with information from their side satisfying exactly the given parameters from the request.

Information Updates – The local node identifies certain patterns within the available event information as important enough to be shared with other nodes from the community or several communities. This way it will assemble an information update in order to populate the significant information.

Besides the need of requesting knowledge from other nodes on top of the well-known concept of populating the messages must provide mechanisms to carry more complex information. This way, one event might be in relation to certain other events which should be carried within the same message and the given relation should be represented as well. Mechanisms should be in place which allow integration of new information in a very simple manner.

Integration of third party event generators

IOIDS has been developed with the idea in mind to interconnect several organisations in order to share their security related information such as intrusion detection audit data. It

is clear that those organisations have security measures and applications in place already in order to protect their infrastructure. IOIDS shall not replace those measures but instead take the available information, normalise it and share it with other nodes. In order to integrate knowledge from other applications (our so-called third-party event generators) the following objectives have to be addressed:

- A modular plug-in mechanism to ease the process of supporting new event generators
- Well-defined and well-documented interfaces to integrate applications
- Adoptable message format to support the transport of very different types of information
- A dynamic backend data repository to make all the information from different sources persistent

Garbling of information

An adequate permission and access control model should be able to protect the local knowledge appropriately. However, due to restrictions applied by the access control certain knowledge will remain on the local node without being populated in order to protect the organisation's assets.

Consequently, it could be very beneficial to publish some information to a certain distribution domain whereby addressing the organisational protection issues by disconnecting the knowledge in question from the identity of the sender. This must be performed on the following two layers:

Anonymising on network layer – The receiver of the message must be prevented from identifying the original sender by examining (source) address information within the network protocols (such as the IP source address). Nevertheless, the receiver must be able to classify the received chunk of knowledge and make sure it was initiated at a trustworthy source within the network.

Sanitising on application information layer – The event information, which is transported within the IOIDS messages, must be rid by all information, allowing inference to its identity. This way, well-known fields with address information have to be replaced by place holders as well as certain pre-defined pattern should be garbled in full-text search and replacement.

The employment of anonymising and sanitising mechanisms will result in sharing a larger set of information without violating the organisation's integrity policy.

Trust and Protection of information

Very sensible information is carried using the IOIDS infrastructure. Thus, strong and reliable protection mechanisms have to be in place in order to satisfy the security needs. It is believed that security within the infrastructure can only be achieved by employment of trust relationships and implementation of security policies. These have to be accompanied by the following issues in order to provide overall security:

- Strong, reliable and easy to configure security mechanisms / security policy – The security policy must be strong on the one hand in order to satisfy the need of protecting the organisational assets. On the other hand, however, it must be easily configurable in order to adopt the needs of each single node.
- Authentication of members – It is fundamental for a secure infrastructure that members may authenticate between each other in a reliable and trustworthy manner.
- Confidentiality and integrity – Messages must not be intercepted by anybody; neither for the reason of gaining information, nor for tampering the content. Furthermore, it must be made sure that a message was sent by the party it claims to be coming from.
- Authorisation and privacy – Local knowledge must be protected in a way that it is only distributed to members, which are intended to read the knowledge chunk in question.
- Decentralised control and local responsibility - All responsibilities within the network should be shared in a peer-based manner; hence, a total avoidance of single points of failure should be achieved. All decisions should be made on the local node, which provides the highest possible degree of trust.

3.4. Terminology

Two major components are essential for this approach, namely the Grid for Digital Security (G4DS) and the Inter-Organisational Intrusion Detection System (IOIDS) (see section 4.4 for a more detailed overview). The former one, Grid for Digital Security, describes all the issues, methodologies and technologies for the subjacent architecture. It is a knowledge based Grid architecture which deals with all issues related to provide and secure the communication channel and provides interfaces for distributing knowledge using this infrastructure (details for G4DS are discussed in chapter 5).

The Inter-Organisational Intrusion Detection System instead is an application running on top of the Grid for Digital Security. It makes use of the provided architecture and provides an

infrastructure for exchanging security related information such as incidents, attack descriptions, information about new attacks in general or related information about countermeasure and the like. (The IOIDS application is discussed in chapter 6.)

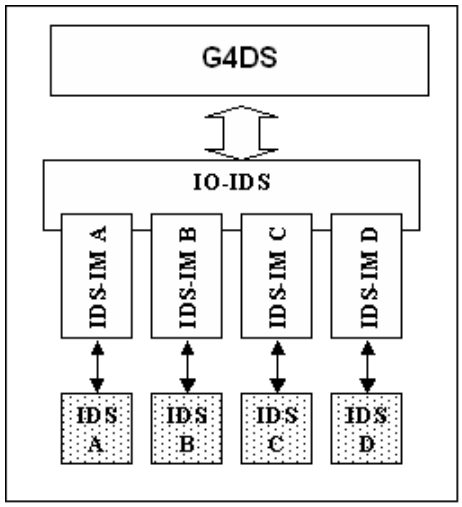


Figure 3.3.: High level description of overall architecture

In fact, the entire system comprises of the following components (check also Figure 3.3):

1. Grid for Digital Security (G4DS) - The G4DS represents the fundamental architecture, which the whole system is built upon. Several issues such as encryption and authorisation are addressed in this module. Due to a decentralised approach users of this module will benefit from a robust and reliable architecture. Trust relationships are built up in this module which will enable applications to make publishing decisions based on the role of the members.

2. Inter Organisational Intrusion Detection System (IOIDS) - The IOIDS is an implementation that utilises the G4DS. It deals with all issues directly related to Distributed Intrusion Detection Systems such as Intrusion Detection message formats and exchange standards.

3. IDS-Integration Module (IDS-IM) - The system will be applicable for a variety of different Intrusion Detection Systems. The integration of the actual IDS is performed by modules which allow an easy plug-in of the different products. At the end of the day, communication can be established between totally different (Enterprise) IDS utilising this Inter Organisations Intrusion Detection System infrastructure.

4. Connected (Enterprise) Intrusion Detection System (EIDS) - Currently, there are plenty of Intrusion Detection Systems available implementing different detection and integration technologies. For this project, no separate IDS will be developed but integration of IDS will be performed. Nevertheless the (Enterprise) Intrusion Detection Systems represent one component of the overall solution.

In the KGrid environment several roles exist with different permissions. Beside the roles, which may be defined on application basis, there are a few KGrid specific ones for making the system work. The following paragraphs give an introduction to knowledge services and Trusting Communities and then list the various roles with their permissions and responsibilities.

3.4.1. Knowledge Services

A knowledge service is the implementation of one specific application within the KGrid topology. It may span over several Trusting Communities or may include members of certain communities, whereby not the entire TC is part itself.

Knowledge Services define the communication (protocols, algorithms) on an application layer base. This means that the knowledge service itself is described using the Knowledge Service Description Language (KSDL), which has been developed as part of this project including the definition of an XML-Schema for KSDL service descriptions.

The Knowledge Service Description maintains information about the following attributes of a knowledge service:

- Unique KGS identifier for the Knowledge Service
- A name for the Knowledge Service
- Current version of the Knowledge Service description
- Optionally, a description
- Information about Service Authorities (SAs) (their identifiers and certificates)
- Information about members and member communities with their identifiers and certificates

An example for a knowledge service description and the corresponding XML schema are provided on the resources CD. The knowledge services are discussed in detail in the chapter for G4DS in sections 5.2.3 and the integration of services in section 5.3.6.

3.4.2. Trusting Communities

Trusting communities are an aggregation of nodes, which agree upon a common purpose and communication protocols. Instead of defining the service properties it defines protocols and algorithms on a lower level of communication. This way, encryption and authentication protocols are agreed upon, which are allowed to be used within the trusting community. Furthermore, agreements about issues such as network protocols are determined here (either to use SOAP, HTTP, SSH, etc.). The responsibility for intercommunity communication including its translation between the protocols is taken by so-called Gateways for Trusting Communities (TCGW) (explained in section 3.4.6).

Each Trusting Community is described in an XML based format with the Community profile. The development of the XML-Schema for this profile has been part of the project. The following attributes for a TC are defined in there:

- Unique TC identifier for the Trusting Community
- Name of the Trusting Community
- Current version of the Community Description
- Optionally, a description for the TC
- Information about Community Authorities (CAs) (their identifiers and certificates)

The concept of communities is discussed in detail in section 5.3.5 within the Grid for Digital Security chapter. Further information about descriptions for communities are provided in section 5.2.3. An example for a community description and the corresponding XML schema are provided on the resources CD.

3.4.3. Members (M)

A member represents any node in the entire KGrid topology. A network node becomes a member as soon as it joins its first community. Members maintain the following attributes:

- In general:
  - Private keys in order to authenticate and sign messages.
  - A public key for each private key to be distributed throughout the different communities and knowledge services.
- For each community it is a member of:

  - A copy of the community description.
  - The identities and descriptions of all CAs of the TC.
  - A copy of a list of all members of the TC. This one will change over the time; in fact, updates for the member lists will be polled from one of the CAs frequently.

- For each knowledge service (KS) it is subscribed to:
  - A copy of the knowledge service description. (KSDL file)
  - The identities and descriptions of all SAs of the KS.
  - A copy of the list of all subscribers for the KS. This one will change over the time; in fact, updates for the member lists will be polled from one of the SAs frequently.

An example for a member description and the corresponding XML schema are provided on the resources CD. Some more information about their processing is discussed within the G4DS chapter in section 5.2.3.

3.4.4. Service Authority (SA)

Service authorities have special privileges for maintaining knowledge services (KGS). For each KGS there are at least two SAs. For small and medium-sized communities every node is intended to be a Service Authority. This supports the approach of avoidance of single points of failure. However, policies different from this one may be supported by limiting a certain set of nodes to carry out the responsibility of a SA. Beside tasks performed by each member of a service, SA have to take care of the following additional matters:

- Signing new members to the KGS. A new member may request its membership to a knowledge service at any SA of a knowledge service. Once the request is granted, the SA populates the information about the new member of the SA throughout the entirety of nodes subscribed to a service, including its identifier and certificates (with the public keys).

- Signing new communities to the KGS. There might be occasions, where it is sensible to add an entire TC to a Knowledge Service rather than single members. This way, any Community Authority of the requesting TC may pass a joining request to any of the SA of the knowledge service. Once the request is granted, the affiliation of the new TC to the KGS is populated throughout the service, including identifier and certificate (with its public key) of the TC.

- Changing roles of members for the KGS. Each member will be equipped with an initial role when joining a Knowledge Grid Service. Depending on the policy for the service this might either be a normal member or a Service Authority. These roles, however, are not static for this member; they might be subject to change later on. Every Service Authority is able to change the status for a member towards a Service Authority. The other way around, however, may not be performed this way. Since all Service Authorities have got the same status (there is no hierarchy), once granted Service Authority statuses cannot be revoked from a member later on.

- Changing the KSDL; hence, introducing new versions of it. During the lifetime of a service, its service description will change several times. Only Service Authorities are able to introduce and propagate a new version of the description.

3.4.5. Community Authority (CA)

After all, the whole KGrid is peer based, thus central instances are totally avoided. However, for the maintenance of communities the introduction of nodes with special privileges is inalienable. (For a very open community one may still define each member as a CA). CAs are defined at introduction time of a new community. New CAs may be added later on by an existing CA. Each TC must have at least two CAs in place. Responsibilities of CAs include:

- Signing new members for the TC. Whenever a member is willing to join a Trusting Community it may request membership at any of the CAs for a TC. Once the membership is granted, the information about the new member together with its identifier and description of the member are populated throughout the community.

- Changing roles of members within the TC. Again, new members of a Trusting Community will get assigned an initial role within the TC depending on the policy of the community. However, these roles are not static and may be changed for any member by any of the CAs of a TC. Downgrading the status from a CA to a normal member is not possible since all CAs have the same status and there is no hierarchy in place.

- Changing the description of a TC. The description of a community will evolve during its lifetime. New protocols or encryption mechanisms may be added and these changes have to be mirrored in the description for the community. Changes to the description are supported by introducing a new version of the Community Description. Each CA is allowed to introduce and propagate a new version of the TC Description File.

3.4.6. Trusting Community Gateways (TCGW)

Within the network of communities, an additional role for inter community communication is required. The so-called Trusting Community Gateways are responsible for passing messages from one TC to another. The passing is restricted to certain rules, which are defined in the TC policy. A TCGW is always defined for a certain pair of communities and for a certain direction. The rules for passing messages are defined in the TC description itself and are administered by the Community Authorities. The TCGW itself is only an executable role. See section 5.3.5 for more details on Trusting Community Gateways and inter community communication.

3.5. Conclusion

In this overview about the objectives and terminology the actual idea of Inter-Organisational Intrusion Detection has been presented and it has been shown, how such a technology fits in modern network infrastructures. Firstly, by explaining three very diverse deployment scenarios for IOIDS the practical usefulness has been presented. By giving a categorised overview about objectives for such an architecture the aims have been laid out for the further research and development process. Finally, a first outline of the upcoming architecture has been given by introducing an essential set of vocabulary, which will be used again and again over the remaining chapters.

With the given information in here the following chapters can present all the details about the IOIDS architecture. First of all, chapter 4 provides a highlevel overview about architecture and implementation and will lay down the procedure for evaluating the project, whose execution is explained in detail in chapter 7. The two remaining chapters 5 and 6 are dealing with all the architectural and technical details for the subjacent communication platform Grid for Digital Security (G4DS) on the one hand and the actual Inter-Organisation Intrusion Detection System application on the other hand.

Chapter 4.

Experiment Definition

4.1. Introduction

So far is has been discussed what the expectations for a so-called Inter-Organisational Intrusion Detection System are. In this chapter, the concept to address those expectations will be introduced. Besides the introduction of the proposed architecture itself an evaluation process is described to examine the capabilities of such a one.

This chapter is divided into four parts. First of all, the concept of IOIDS is put into IT security context and the potential for enhancements to be gained with such an approach is outlined. Afterwards, an evaluation methodology is described, which compares IOIDS with similar or related approaches by the processing of available publications and documentations. The third section is dealing with implementation issues of IOIDS and explains briefly the components of the architecture including references for more detailed information. Following this the last section draws attention to the practical evaluation process for IOIDS and, this way, defines the way in detail, of how to carry out an experiment in a laboratory environment.

4.2. Overview

Recent surveys and observations have shown that security measures in general and intrusion detection systems particularly have not yet achieved the results they should. It is commonly agreed that the problem is not grounded in the missing of audit data in the first place, but much more in the processing, generalisation and distribution or exchange of the processed results. Practically, there is too much data presented to the user in an unstructured manner so that they, in the end, cannot take much advantage of intrusion detection measures in place.

It is believed that the potential of improvements for intrusion detection technology may be categorised under the following three topics:

- Improvements for the sensors, which results in higher quality and more exact and

complete audit data at the gathering components

- Enhancements in the view of correlation and abstraction of data in order to identify similar attack patterns and reduce the amount of data in progressive stages

- Distribution of data among several peers in order to identify distributed attack patterns and in order to tackle the security problems in IT networks as a unit rather than by each party in an isolated manner

Intrusion detection systems are not a completely new technology and since their introduction about two decades ago new detection mechanisms have constantly been developed and deployed. It can be stated that a vast amount of research and development is put into this area, which results in ever improving mechanisms for category one.

The need for generalisation and correlation of ids audit data has been identified soon after wider deployment of intrusion detection system in real-world throughout the Internet. Security analysts were facing ever growing amounts of data – the reporting of single alerts or a simple counting or ordering by certain attributes could not support very well with identifying attacks in the network. For about one decade researchers and industry attended to this problem and initiated a number of projects and approaches, which employ very different technologies such as data mining, regular expressions, object orientation or neural networks for generating a more abstract view on the data. It is believed that more efforts will be put into this topic over the upcoming years and that future intrusion detection system deployments will benefit significantly from enhancements in this area.

The last category with the problem of distribution of ids audit data has started to gain attention over the last decade only. Firstly, intrusion detection system vendors became aware of the requirement to correlate ids event data from several location within the local network, which is usually carried out in a hierarchical manner with deployment of a central event data base, where all event data from all sensors (usually passed through certain proxies) ends up. These modifications have shown significant improvements in the usability of ids audit data. However, to date the distribution of this data is forcefully kept inside organisations – in the authors belief due to a lack of concepts to protect this sensitive knowledge appropriately and a way to establish trust relationships with parties in a simple manner and exchange this information between them based on these trust relationships. A major issue for trusting an approach is, in the author's belief, the fact to maintain control about the level of trust and the information shared on the local node.

The following three sections will explain the problem of intrusion detection technology in the overall context of information technology. Afterwards, shortcomings of currently available

approaches are presented and, last but not least, a list is given, which mirrors the potential for enhancements for distributing intrusion detection audit data in the view of the author.

4.2.1. Problem in IT security context

In the early years of the Internet developments were not carried out with drawing much attention on security issues for the transmission of data. Consequently, nowadays there are a high number of protocols in place in the Internet, which were not designed to be used for security sensitive contexts. As a result, a number of high level protocols have been established on top of those in order to implement security as kind of a work around. Additionally, tools and applications have been developed and deployed with the purpose of controlling and monitoring the networks and identifying as well as reporting malicious or anomalous behaviour.

An early approach for protecting networks was the deployment of firewalls. In the early years of firewalls, very simple technologies such as packet filtering regarding network addresses or applications (port numbers) could raise the barrier for an intruder to reach their aims. With deployment of those technologies, however, attack patterns were developed to evade them and more sophisticated firewall approaches were brought into being such as application layer firewalls, which perform inspection of the network traffic and attempt to identify malicious patterns.

An ever growing thread in the Internet is the spreading of viruses and trojans. A major problem with these threads is their non-recognisability by firewalls as they are making use of communication channels, which are usually not blocked by firewalls (such as email or HTTP). Nowadays, the most common way to tackle the problem of viruses and trojans is the deployment of anti-virus software. This is performed both, on each node in the local network (desktop anti-virus) and on central instances such as central virus scans on email servers. These technologies support well with countermeasuring viruses, however, the signature based approach always requires a priori knowledge about new viruses, which leaves a certain time gap for the virus to spread between its release and the update of the signature database in the antivirus software.

More problems are based on the theft of information, usually carried out using technologies like eaves-dropping or man-in-the-middle attacks (leaving behind the huge amount of theft carried out by utilisation of social engineering technologies). Employment of authentication and encryption mechanisms based on public key infrastructures are looking to overcome these problems.

The presented countermeasures are of course only a collection of very popular candidates

(see also section 2.2 in the state of the art chapter for more information). The problem with this collection, however, is the lack of a reporting mechanism, which is able to provide information about the status of the network and its entities whenever none of the existing mechanisms was able to identify or prevent malicious activities. At this point the approach of intrusion detection kicks in - due to its passive nature it can observe and evaluate activities in the network or on a host without being identified by a potential intruder.

4.2.2. Shortcomings of available solutions

As previously mentioned, intrusion detection has occurred in the context of information technology quite a few years ago already. Many efforts have been put on the improvement of this technology over the last years; however, with enhancements in the detection mechanisms the intruders align their attacks and behaviour to the new environments and again and again they find ways to evade novel technologies. Practically, the modifications and additions for intrusion detection (and often for IT security in general) are rather reactive than proactive; meaning that the defenders side is most of the time a few steps behind the intruder. Consequently, major topological and structural changes should be applied in order to turn around this relation.

In particular, the following list names some major issues identified for currently deployed intrusion detection system infrastructures:

- Existent distributed intrusion detection systems are usually structured in hierarchies or, even worse for the described scenarios, in centralised approaches and do not scale very well for deployment in global environments.

- Organisations maintain certain relationships between each other. Based on experiences with certain parties a member decides how much to trust information from this party. An intrusion detection system should be able to map these relationships into its system in an easy and straight-forward manner rather than introducing a new concept, which should be applied to the relations between the organisations.

- The deployment of distributed security measures is often accompanied by handing over responsibility or decision making potential to certain (often central) instances. (Very good examples are anti-virus databases, which depend on frequent signature updates from the anti-virus software vendor). Systems should enable parties to benefit from collaborations in distributed intrusion detection without withdrawing power from them, instead responsibility should be kept on the peer.

- Some approaches for distributed intrusion detection may scale quite well (even for global environments); however, these are very restricted to certain features of the overall feature set requested from intrusion detection systems.

- The Internet is a world-wide network of inter-connected nodes. Consequently, isolated IDS are not able to achieve satisfying results due to their lack of examining distributed attack relevant information.

Over the last years, the list of enhancements towards distribution for intrusion detection resulted in improvements of the overall results. It is believed, however, that the potential of this idea has not yet been reached due to the lack of exchanging the gathered information on the highest level; meaning between the organisations.

4.2.3. Potential for enhancements

Taking all these shortcomings into account, it is believed that there is great potential to maximise the benefits of employing distribution features in intrusion detection systems. In section 3.3 the objectives for inter-organisational intrusion detection have been discussed in detail; however, the following list shows a selection of potential improvements in comparison to currently available (distributed) intrusion detection systems:

- A grid based network infrastructure, which is completely based on a peer-to-peer topology comes with significant improvements in the view of availability.

- Establishment of trust relationships between parties and mapping of those into the software accompanied with classifying the knowledge ends up in an easily manageable approach for protecting local information.

- The integration of many kinds of intrusion detection systems from different vendors enables the overall system to benefit from the advantages of all detection and correlation mechanisms. Additionally, a well-defined standard-based integration approach and interface allows easy integration of new detection technologies.

- A distributed intrusion detection system approach should be highly scalable. Consequently, it may be applied to a wide range of deployment scenarios ranging from small local deployments up to Internet wide distribution of intrusion detection audit data.

4.2.4. Experiment relevance

The upcoming approach is requested to fully support the hypothesis of finding a solution for exchanging intrusion detection audit data across organisational boundaries in a secure and non-reputable manner, while maintaining commercial confidentiality. The way to evaluate the outcome can only be performed by going through the following steps:

1. The available objectives must be examined to be a correct and complete list of features for the defined problem.

2. IOIDS must prove to satisfy all the objectives, which were worked out for such an approach.

3. It must be made clear that there is not such an approach available to date, satisfying the list of objectives already; meaning, that there must be a clear proof of contribution to science by IOIDS.

The analysis has to be carried out by breaking down the objectives into atomic pieces, which can be directly examined and measured for different approaches. As the design and realisation of a proof-of-concept implementation was part of the project, practical experiments shall be carried out in order to evaluate its behaviour in real-world scenarios. Comparison with state-of-the-art approaches within the DIDS context as part of these experiments shall prove the superiority of IOIDS over available solutions.

How far IOIDS is able to satisfy all the given ideas is examind over the following pages. After introducing a number of related or similar projects in section 4.3 an introduction to the idea and architecture for IOIDS is provided and linked to the corresponding chapters for the involved components. The definition of the evaluation process in section 4.5 shows, how IOIDS shall be deployed in experiments first of all, and how the corresponding results shall be used for an analysis afterwards.

4.3. Comparison

In section 3.3 the objectives for an approach to implement intrusion detection communication between organisations have been discussed in detail. To date there is not yet known any approach addressing the entirety of these objectives; however, ideas, applications and approaches have been published, which are related or similar to the IOIDS idea or address a certain subset of its objectives.

Only a breaking down of the objectives into single features and the direct comparison against similar projects on these attributes enables us to evaluate the features of IOIDS. For these reasons the following two steps had to be performed:

1. A number of available projects, applications or approaches have to be chosen to compare features. This choice must be performed in a way that different entities cover different sets of features so that a collection of very diverse projects from within the context will be explained and evaluated.

   Some of these approaches only exist in theory and, consequently, IOIDS may only be compared against them in the view of matching features. Other approaches however come with an implementation and may this way serve as candidate for practical evaluations and performance measurements or benchmarking.

2. A structured collection of broken down features or attributes have to be worked out. They should be grouped regarding their meaning and a description for each of them must explain its meaning in the context of the project.

The following two sections discuss these two steps in detail. Firstly, a number of projects will be presented with some meta information for each of them. Afterwards, objectives are broken down into textual groups, which the final low level attributes will be assigned to.

4.3.1. Representatives

In order to measure and evaluate the features, attributes, benefits and shortcomings of IOIDS a selection of similar or related products and approaches have been chosen for comparison. A short introduction of each of them with their context and the reason of their choice is following.

In the context of distributed Intrusion Detection the following approaches are considered to be worth for evaluating the Inter-Organisational Intrusion Detection System approach:

Snort – Snort is the most popular open-source network intrusion detection system available these days. Regarding to the authors snort claims to be the de-facto standard for intrusion detection / prevention. (Snort (2006))

Snort's development has been initiated by Martin Roesch in 1998 and since then it has ever grown in popularity. Due to its modular approach with its preprocessors and output plug-in mechanism it can and has been deployed in numerous, diverse employment environments.

Snort in its original nature is not considered to come along with distribution features; however, due to its wide employment as part of such scenarios it has been chosen as one candidate to measure IOIDS on.

SnortNet – In the year 2000 Snort was extended by distribution features and a project named *SnortNet* was born. For some reason the development of SnortNet has never been taken much further and besides the initial efforts in 2000 no more versions have been released.

The author of SnortNet added distribution functionality by the use of Snort's output plug-in mechanisms and ended up with an architecture consisting of network sensors, proxy daemons and a monitoring console, which together form up a centralised, hierarchical architecture (Fyodor (2000)). Great attention has been drawn to the employment of well-known standards, so that the Intrusion Alert Protocol (IAP) (Gupta et al. (2001)) as transport protocol and the Intrusion Detection Message Exchange Format (IDMEF) (Curry et al. (2002)) for Intrusion Detection Messages have been utilised.

Due to its direct link to the actual Snort IDS it proves significance for comparison against IOIDS. Although slightly outdated, it is believed that it can serve well with its distribution features for performing the analysis.

Prelude – Since Prelude (Vandoorselaere et al. (2004), Prelude), another Open-Source Intrusion Detection System, is coming with both network as well as host intrusion detection features, it defines itself as a Hybrid Intrusion Detection System. It was brought into being in 2001 and since then ever grown in popularity as an alternative to the Snort IDS. Last but not least the changes in the licensing model of Snort (and thereby especially its signatures) have boosted the importance of this IDS.

Prelude understands itself as a framework to integrate audit data from a variety of sources rather then a plain intrusion detection system. Therefore it provides a collection of libraries to be used when integrating an application with the framework. The final intention is to collect data from a variety of diverse data sources into a central repository in order to benefit from the advantages of all of them. Besides the libraries a set of components such as event collectors and normalisers, log analysers and a syslog integrator as well as an IDS management console (Prewikka) are provided to build up the IDS topology.

Prelude offers useful capabilities regarding the distribution of IDS event audit data. Fortunately, it does not only exist as a theoretical approach but also the software is available itself; consequently, it can serve very well in practical analysis and benchmarking evaluations (see section 4.5).

AirCERT – Automated Incident Reporting (AirCERT) (Cert/CC (2006), Danyliw et al. (2003)) is a high-level, scalable distributed system for sharing security event data among administrative domains. In contrast to the previously presented approaches it is intended to exchange IDS event data between organisations rather than inside them. Due to its scalable approach it is able to transport any data ranging from network IDS raw data up to entire incident reports.

AirCERT's architecture is comprised by normalisers and sensors, collectors, the *ACID* Analysis Console for Intrusion Databases and publishers and incorporates standards such as the Incident Object Description and Exchange Format (IODEF) (Demchenko (2003)), the Intrusion Detection Message Exchange Format (IDMEF) (Curry et al. (2002)), and the Simple Network Markup Language (SNML) (Center (2003)). Additionally, a set of libraries is provided which allows reuse and integration of other applications with the AirCERT architecture. AirCERT is still under development and up-to-date versions of the software packets may be downloaded from its website (Cert/CC (2006)).

AirCERT qualifies as a candidate for evaluation purposes of IOIDS due to its approach of exchange ids event audit data across organisational boundaries. Although it is not stressing the approach of trust relationships between these organisations very much it has quite a commonality in the original idea for the approach with IOIDS.

Cooperating Security Managers – In the year 1996 White et. al. put the technology of intrusion detection with its Cooperating Security Managers (CMS) (White et al. (1996)) in the deployment context of wide area networks for the first time. The peer-based approach looks to address the gap in topology between distributed intrusion detection systems at that time (which were mostly hierarchical) and the actual network topology (which does not tend to be hierarchical for large scale networks such as the Internet).

The CMS topology is made up (on each participating peer) of a detection component and a security manager, which processes information from both the local and remote detection components. Additionally, it is equipped with a graphical user interface in order to allow user feedback and interaction. Last but not least shall a so-called intruder-handling component take care of countermeasure procedures. The major limitation of this approach is its exclusiveness of employment for user-tracking throughout networks in order to allow the assembling of user profiles.

Although very limited in its functionality concerning IDS features (from the today's point of view) the CMS with their peer-based approach and autonomously acting entities show similarities to the Inter-Organisational Intrusion Detection System. It will

be interesting to compare the two regarding their reliability and robustness features.

AAFID – The beginning of research on the Autonomous Agents for Intrusion Detection (AAFID) (CERIAS (2000), Balasubramaniyan et al. (1998), Spafford and Zamboni (2000)) dates back to 1998 and established a very light-weight approach for distributed intrusion detection. A prototype (AAFID2) was released to the public in 2000, however, afterwards, the project was discontinued.

AAFID looks to implement the idea of intrusion detection by employment of many small, autonomous entities instead of employing a monolithic design. Thereby, its major purpose is to provide the framework; meaning the communication platform the agents may communicate through. Its overall architecture is made up of the agents themselves, transceivers (considered as the communication interfaces for the hosts), monitors (the highest-level interfaces in AAFID) and, finally, the user interface for presenting data and interacting with the system.

The time AAFID was developed a variety of projects were researching into light-weight approaches for distributed intrusion detection (see also section 2.4.5) - not all of them carried the word agent in their name, but after all the initial ideas were similar to the one of AAFID. None of these project seems to be continued to date and the Autonomous Agents for Intrusion Detection is considered as the furthest developed of the idea; consequently AAFID represents a concept here in general when measuring capabilities of IOIDS.

GIDA – Quite recently (2002) Tolba et. al. started researching into the idea of intrusion detection technology applied to grid topologies and a series of publications (Tolba et al. (2002), Tolba et al. (2005a), Tolba et al. (2005b)) have ended up in an approach called *GIDA – Grid Intrusion Detection Architecture*. GIDA looks to apply IDS technology on existing grid architectures rather than creating a grid architecture to support distributed intrusion detection.

The GIDA architecture comprises of Intrusion Detection Agents (IDA), the gathering components, and Intrusion Detection Servers (IDS), which are in charge of analysing the data and cooperating with other IDS in order to detect intrusions. An unique feature of GIDA is the introduction of administrative domains, which supervise the deployment of IDAs and their registering with IDSs. It integrates with the established standards for grid services and, consequently, is built on top of the Globus Grid Security Infrastructure (GSI) (Welch et al. (2003), Foster et al. (1998)).

GIDA shows a number of similarities with IOIDS since it also looks to combine the

two technologies distributed intrusion detection and grid computing in one approach, although carried out from a different point of view. It is a very up-to-date approach, which is still under development. The authors have carried out some experiments and performance measurements on GIDA, which enables us to mirror some of its features against IOIDS capabilities.

4.3.2. Overview of desired features

In general the requirements for implementing inter-organisational intrusion detection system communication have been discussed in the objectives in section 3.3. In this section we are looking to break down these objectives into single entities – lowlevel features to allow direct comparison between approaches.

Similar compilations of features for (distributed) intrusion detection have been created (Liesen (2002)) and could this way contribute to assembling this overview. In order to apply a content depending structure to the list of features, groups have been created, which the attributes are assigned to. This way, each attribute is member of one of the following groups:

Intrusion Detection – Relating to the intrusion detection capabilities of the approach.

Distribution – Options for distributing knowledge throughout the connected domains.

Security and availability – All security issues ranging from authentication and authorisation issues to resistance against attacks targeting the system in question itself.

Extensibility and collaborative options – Features for integration of the system in questions with third-party products and ways of extending its functionality.

In the following sections each group is presented with its representatives; the meaning of each attribute is explained in the context of inter-organisational intrusion detection.

Intrusion Detection

For the category concerning *intrusion detection* facilities as those, the following entities have been chosen:

- Support for different detection types: Currently, there is a variety of different approaches available for detecting intrusion. Many attempts have been undertaken to categorise these different technologies (also see section 2.3); it is commonly agreed to

separate the two groups network intrusion detection (NIDS) and host intrusion detection (HIDS). In order for an intrusion detection system to work efficiently, it should support the implementation of both of them.

Further separation is performed regarding the way malicious behaviour is identified - popular candidates are signature-based or anomaly detection based approaches. Implementation of both them is considered as beneficial for the performance of IDS.

- Completeness of detectable attacks / patterns: Certain intrusion detection system implementations only support detection of certain kinds of attacks. A complete solution, however, should employ a combination of technologies in order to cover the entire set of attack patterns.

- Correlation of event information: Certain incidents may only be identified if knowledge from several low-level events (possibly gathered in different locations) is correlated and analysed. An IDS should be able to perform (or at least support with) this task in order to identify for example (locally) distributed attack patterns.

- Stateful inspection: Traffic on current networks such as the Internet is transmitted in packets whereby large amounts of data is broken down into smaller pieces on the sender side and assembled on the receiver side. Certain attack patterns take advantage of this circumstance and span malicious code over several network units (such as IP fragments or TCP segments) in order to bypass employed incident detection procedures. As a result, an intrusion detection system should be able to reassemble this traffic in turn in order to analyse it and identify malicious patterns. This feature should range from IP fragment reassembling (ISO OSI layer three) up to application layer traffic reassembling (ISO OSI layer seven).

- Reporting and user interaction: Detecting incidents of the network is only one part of the job of an intrusion detection system. The second responsibility is the reporting of incidents to the user. This should be supported for a variety of transmission medias such as SNMP traps, email, pager or syslog.

  Furthermore, an intrusion detection system should be able to provide structured summaries of gathered information in form of reports. After all, alerting and reporting must be carried out in a way and at a level of complexity that a user is not overwhelmed with alerts but still gains enough information in order to react and countermeasure appropriately.

- Response and countermeasure: On top of detecting and reporting malicious behaviour an intrusion detection system should support with the process of initiating reactions in reply to certain activities (either automatically, semi-automatically or by manual initiation of the user). For this task it often needs to interact with other components of the network infrastructure (such as firewalls, routers, domain controllers, etc).

- Integration of variety of products from different IDS vendors: Whenever an approach is looking to make use of third party intrusion detection system facilities instead of implementing them itself, it should provide mechanisms to integrate with a variety of products of different types and vendors.

Distribution

In the category *distribution* the following features are thought to be measurable:

- Real time capabilities: The time delay for the distribution process of information is very critical due to the requirement of initiating appropriate reactions and countermeasures as soon as possible. Although a distribution of information in literate realtime throughout the networks seems in practise impossible due to latencies in subjacent network topologies the delay must be kept on a minimal time frame.

- Scalability: For wide deployment in different topological infrastructures the approach should be designed in an adoptable manner regarding its sizes or frequencies. This requirement addresses several entities within the infrastructure such as the number of network nodes, the number of domains as well as the number of events and, thereby, the amount of network traffic exchanged.

  If the approach in question does support some kind of domains or communities, it should also support a scaling of this feature, ranging from very small communities up to large communities and from a small number of communities up to a large number of communities with keeping the network traffic at a reasonable level. For example, different layers of abstractions for the event information for different layers of communities could be requested if a deployment scenario looks to implement a topology like three stage approach (hierarchical), where low level, detailed event information is exchanged locally in small, local domains; more abstracted and correlated information is exchanged on a middle layer and rather general and very abstract event information is generated and exchanged in a top-level layer (just in order to take a three layer approach as an example).

- Dynamics: Distribution domains for the event information are not static; instead they are likely to change over time: new nodes need to be added to domains, nodes may want to be removed from a domain or the status of a node shall be modified. Furthermore, if inter-domain communication is supported the evolvement of new domains will affect other domains as well. These organisational processes should be supported by the management system of the approach either fully or semi automatically in order to keep manual intervention on a minimal level.

- Mapping and mirroring of underlaying network topologies: In the explanations for scalability the intention and likelihood for deployment of different scenarios and topologies has been mentioned. These topologies might not only be different regarding their size but also regarding their organisational and technical topology.

  The intrusion detection system should assist with the mirroring of this network topology in order to realise a mapping between them. This allows the user to trace events more easily and assess their impact regarding their locality.

- Duplicate data identification and avoidance: In an environment, where a number of nodes are processing data, deriving information and passing on information (especially in peer based approaches with support of domains), equivalent or similar pieces of data may occur in certain circumstances. Measures must be in place to either avoid this behaviour a priori or identify duplicate chunks of information and initiate appropriate reactions.

Security and availability

The features within the category *security and availability* range from very low level features such as authentication and encryption up to trust relationships and their implementation. In detail this category is made up by the following list of requirements:

- Authentication of members: Strong authentication mechanisms must be in place at all locations, where communication between two parties is taking place and may be intercepted for reasons such as spoofing attacks. This does not only apply to communication channels between two different peers but also between the components of the local infrastructure of a single peer.

  In order to assist this process up-to-date authentication mechanisms such as asymmetric key authentication (public key infrastructure) should be implemented. Furthermore, it is requested to authenticate the communication in both directions, stating that both sender and receiver must authenticate at the beginning of the transfer.

- Encryption: All communication channels within the overall topology must be protected against unauthorised access using strong encryption facilities. This is due to the sensitivity of information exchanged using the infrastructure. For performance reasons (after the initialisation process of the communication) symmetric keys should be employed for encryption purposes.

- Integrity: The entire idea of distributed intrusion detection bases on the trust given to the information one receives from another side. A receiver must be totally sure about the fact that the information present has been sent from the source it claims to come from. The employment of digital signatures on the sender's side and the message validation using this signatures on the receiver's side is commonly agreed to be the up-to-date technology for these purposes.

- Public Key Infrastructure: Public key infrastructure is actually considered as a technology rather than a feature. However, due to the low level of features presented here it is considered to give information about the security level of an approach and, this way, assists well in the comparison process.

- Single sign on: In modern grid environments it is unacceptable to request a user for credentials again and again if another resource is required. Credentials should be delegated between the peers inside the domains in order to avoid this kind of interference. If required and possible, credentials should be derived and exist in generations in order to restrict the abilities for a derived or delegated credential (such as its life time or the ability to delegate the credential on further).

- Decentralised Control: In the objectives it has been stated that for availability purposes a distributed IDS approach should employ a peer based network topology and, this way, guarantee a total avoidance of central instances. There must not exist a single feature within the overall topology which may be powered-down by being successful in disconnecting a single node or a certain set of nodes.

- Possibility of integration of different protocols and algorithms: Recent global developments with modifications in legislation in a number of countries do not allow employment of certain PKI algorithms in certain places. Different algorithms should be integratable easily and the selection of usable mechanisms must be configurable for the infrastructure - preferably for the domains of the topology.

  Similar problems occur when drawing attention to the employment of communication protocols for information exchange between the peers. Certain configurations within

organisations do not allow utilisation of a certain set of communication protocols - especially when intended to be used across organisational boundaries, saying over the Internet. Consequently, next to the demand of easy integration of communication protocols, these ones should be easily configurable and selectable for certain network domains.

- Local responsibility: The attitude of the idea discussed here is a peer based approach which allows every peer within the network to establish trust relationships with members of their choice. For these reasons it is very important that the responsibility for making decisions resides locally with the members. Employment of any central instance, which makes decisions on behalf of a node would undermine this concept. Consequently, (although domain or group policy must be in place and enforced) it is beneficial for the approach to leave it entirely up to the local member to make decisions about how to progress information from which source and which information to share with which distribution domain.

- Credential mapping between administrative domains: The requirement for employment of administrative or organisational domains or groups has been discussed already. However, the opportunity to share knowledge across these domain boundaries requires a mapping of credentials between such. A mechanism must be in place, which enforces and secures the authentication and authorisation across boundaries in a transparent manner for the peers.

- Avoidance of single points of failure: In the feature *decentralised control* the basic concept for distributing all responsibility throughout the infrastructure has been described. This attribute stresses the idea more on the ability of the overall system to continue its work even in case of interruption of certain members or services. It is illusive for modern network topologies to maintain a hundred percent availability of all entities; the more it becomes important, however, to keep the overall infrastructure running when parts of it are interrupted or delayed.

- Authorisation and privacy: As much as the information must be secured when travelling inside the networks, it must also be protected for the access on the local node. Access control mechanisms must be in place which enforce that information may only be accessed by parties, which are supposed to do so. Beneficial for these issues is the introduction of several levels for security or protection for different chunks of information; meaning, that event information of different classification are regardingly only accessable by different sets of members or domains.

- DoS and DDoS resistance: As much as the intrusion detection system infrastructure in question is supposed to support with identification and reaction for incidents it needs to be protected against attacks directed against the system itself to be of use. Denial of Service (DoS) resistance may usually be achieved on the local node itself by error proof implementation and employment of fallback and recovery procedures. Distributed Denial of Service (DDoS) attacks in contrast may hardly be addressed and countermeasured on the local node only; instead the overall infrastructure should have measures in place for employing fallback strategies as well as resolving and countermeasuring this kind of attack patterns.

- Resistance against internal attacks: A waste amount of attacks are currently carried out by internal intruders. For a grid system it is of course not particularly easy to define the domain *internal*. It will finally depend on the infrastructure and organisation of the approach; however, measures should be in place to address malicious behaviour from trusted parties.

Extensibility

In the view of *extension and collaboration* facilities the following features are thought to be of interest:

- Integration with other IDS products: No single Intrusion Detection System product is able to integrate all features existing in the field of intrusion detection. For optimal results, however, the employment of many different detection technologies and approaches is thought to be very beneficial for improved results. The opportunity to integrate third party products with the approach can assist very well with this feature.

  Providing a standard and well-documented interface for other products for their integration is the first step towards addressing this issue. Furthermore, the number of products that have been integrated with the approach already indicates its usefulness.

- Standards for communication protocols: These days people become more and more aware of the need for employing commonly agreed and documented standards in the field of information technology in order to allow interaction with other approaches or applications and guarantee the opportunity to access the information and carry out interactions even in long term future.

  Whenever an application provides an interface for third party approaches it must make sure that this interface is based on well-defined and commonly agreed standards. Ad-

ditionally, the approach should also employ standards for communication internally between its components.

- Modularity: For reasons of re-usability and ease of maintenance software must not come as one big chunk, but should be broken down into smaller pieces regarding their functionality. On the one hand, the maintainability of the single components is better than the one for the overall complex system. On the other hand, broken down modules encourage the utilisation of certain components for other projects.

- Configurability: It has been stated that this approach should be adoptable for a variety of diverse deployment scenarios. This requires facilities for configuring the application in order to suit the local needs. Configuration facilities should be straight forward and well documented. Furthermore, sample configurations should be provided.

- Heterogeneity: Deployment scenarios do not only differ in the view of size of its components but also in the view of their nature concerning platforms, the components are running on. Consequently, it is beneficial for the approach to support of a wide range of different operating systems, of different data base management system (whenever a database is employed as data repository) as well as hardware platforms.

- Integration with locally employed security measures and systems: An intrusion detection system does not provide security for an organisation in an isolated manner. Instead it has to integrate with other security measures and infrastructural conditions, which are either in place already or planned to be employed.

- Integration with locally deployed central management console: As previously stated the intrusion detection system measures must integrate with other security measures on the node. In optimal conditions this also includes the integration of the intrusion detection system with a local management system. This local management system allows the user to gather information from all the components of the security infrastructure and supports bidirectional interaction with them.

4.4. Execution: Design, Architecture and Implementation

The idea of Inter-Organisational Intrusion Detection appears to be very complex and has to address a vast number of requirements as worked out over the previous chapters. It is indispensable to break down the overall system into components in order to address requirements in working units. As shown in figure 4.1 the following components are present in IOIDS:

G4DS - The subjacent communication platform – All low-level communication features for the system have been encapsulated in a module named *Grid for Digital Security* (G4DS). An introduction to G4DS is provided in section 4.4.1.

IOIDS - The actual DIDS component – On top of G4DS the actual platform for exchanging intrusion detection event data is performing its job. An overview for the actual *Inter-Organisational Intrusion Detection System* is presented in section 4.4.2.

IDS-IM - The modules to integrate third-party event generators – As previously stated, IOIDS does not perform detection of intrusions itself, instead it comes as a high-level approach for exchanging knowledge gathered by other event generators. Consequently, mechanisms must be implemented to integrate information from these applications into IOIDS. The idea for these *Intrusion Detection System Integration Modules*, which allow adoptable integration of applications, is explained in section 4.4.3.

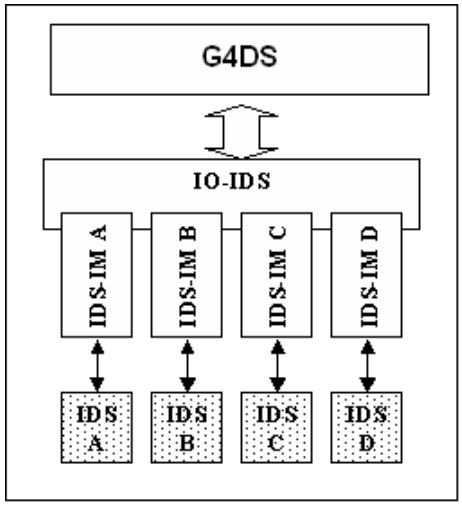


Figure 4.1.: High level description of overall architecture

Practically, the connected audit data generating applications (such as intrusion detection systems or event loggers such as syslog or Windows event log) are acting as gathering components, which report their information in a well-defined way through a plug-in-mechanism to the Inter-Organisational Intrusion Detection System core. The IOIDS in turn is prepared to process this information and initiate corresponding actions including the option of distributing the new information to other members of the IOIDS network. In case of a request for distribution, a G4DS connector, which is part of IOIDS, is receiving the piece of information together with data about the distribution domain and the options for protecting the

knowledge from the IOIDS core and passes this on to the G4DS system, which it maintains a connection to since boot-up time. The G4DS in turn processes the distribution and protection information and with the information about the members of the network it is able to deliver the data to the required parties in a secure and protective manner.

The issues in each of the three components are discussed in more detail over the following sections. The subjacent communication platform G4DS is introduced firstly, for each of the remaining components IOIDS and IDS-IM follows a section afterwards.

4.4.1. Subjacent Communication Platform

As subjacent communication platform a secure and reliable infrastructure had to be developed, which addresses the issues concerning low-level communication such as authentication and non-alteration. The requirement for high scalability had to be kept in mind in order to allow deployment in local as well as large-scale environments.

A further important issue is the re-usability of this infrastructure for other projects. A standardised and well-documented interface should be provided for applications in order to allow easy integration. Configuration and initialisation for the network with their domains is also important.

Responsibilities

From the list of the overall objectives for the entire infrastructure the following items are assigned to the subjacent Grid for Digital Security:

- Establishment of a reliable communication platform
- Securing of all communication using strong authentication and encryption mechanisms
- Support of domain concepts for supporting groups within the overall network including information exchange between those ones
- Providing a platform for authorisation

Implementation

Grid for Digital Security has been implemented by coming up with a grid architecture designed for knowledge exchange, which is based and built upon well-known standards in grid developments. Due to the difference in the nature of knowledge grids and traditional computational grids, which share more physical resources rather than knowledge, the reference

architecture and implementation (Open Grid Services Architecture (OGSA) and Open Grid Services Implementation (OGSI) with Globus Toolkit) could not be employed directly for G4DS. This approach was far to heavyweight and finally focused significantly on sharing computational resources. However, there are still a large number of similarities between computational and knowledge grids and much expertise from experiences in and around OGSA and Globus could be integrated into G4DS.

In practise, the architecture and implementation for G4DS is marked by the following attributes:

- Each node within the network is called member and is equipped with a unique member id. Several low-level communication protocols are supported; G4DS provides an address resolution between their addressing scheme and G4DS ids.
- The entire G4DS network is divided into so-called communities. Communities may overlap and inter-community communication is supported depending on the community descriptions.
- Security issues are addressed by establishing a public key infrastructure.
- Availability is provided by employment of a strict peer-to-peer infrastructure.
- Applications are integrated into G4DS as so-called *Knowledge Services* and are equipped with a unique service identifier.
- All entities within G4DS (members, communities and services) are defined using well-defined XML based descriptions.
- Access control issues are resolved using an access control matrix.
- G4DS runs as a service on linux / UNIX machines and applications may connect to it using a client library, which encapsulates the FIFO communication and rendezvous process. Client applications have to authenticate using asymmetric keys.

Chapter 5 discusses in very detail the design and implementation of Grid for Digital Security. Besides a general overview about how to address aforementioned objectives in the design stage (5.2) it also discusses implementation issues and how to integrate with connected applications (5.3).

4.4.2. Event Exchange Mechanism

The Inter-Organisational Intrusion Detection System component is the heart of the overall system and, this way, in charge to process new information and make decision about corresponding reactions including the option for distribution. These reactions should be easily configurable for the user of IOIDS.

It must implement the interface to connect against G4DS. On the other hand, it should be able to integrate a number of third party audit data generators. For these reasons, it has to provide a standardised and well-documented interface in the first place, and, furthermore, needs to provide modules to support a number of those products from the start on.

Besides the integration of locally deployed third party event generators it must process, handle and store information from other peers of the IOIDS network. A central data repository has to be put into place, which in turn must be able to deal with event data of applications of very different types by maintaining information about the origin and source as well as protective attributes of the knowledge chunk.

Responsibilities

The following list of broken down objectives may be directly assigned to the IOIDS component of the overall topology:

- Central storage of event data
- Processing of information and decision making based on user preferences
- Classification and protection of audit data
- Integration and normalisation of third-party event data
- Implementation of authorisation mechanisms
- Mapping and processing of G4DS credentials and identifiers

Implementation

The Inter-Organisation Intrusion Detection System component has been implemented with the concept of G4DS as communication platform and is this way directly depending on such. It provides mechanisms and configurations for a event database, which serves as a central repository for all event data from any type of application. Communication protocols between peers have been developed with focus on standards for exchanging IT related incident information.

In particular, the IOIDS implementation is described by the following features and attributes:

- An XML based IDS specific event database has been employed as central audit data repository. This approach normalises all event data into a common event core; remaining information is maintained in application specific database extensions.
- The data engine bases its decisions on policies, which are given in an XML syntax by the user. Sample policies for the data engine have been created.
- Information within IOIDS is classified using 10 different classes. These classes allow control of distribution to certain domains.
- Distribution domains for IOIDS may be either a single member, an entire community or all known members of a service.
- Besides the distribution of new information a node may also request information from other nodes by defining certain parameters and sending a knowledge request.
- All communication between IOIDS peers is performed using XML encoded messages.

In chapter 6 all the details about the IOIDS design, architecture and implementation are provided. High-level information about issues such as database backend implementation and communication protocols (section 6.2) are discussed as well as low-level implementation issues such as G4DS connectivity and internal data structures (section 6.3).

4.4.3. Data integration from third party event generators

IOIDS itself is an approach for the purpose of exchanging intrusion detection event data rather than gathering this data. Consequently, IOIDS must be able to integrate and process information from other so-called third party event generators. For this purpose a modular mechanism has to be put into place, which allows easy integration of new applications.

Practically, there are two ways of integrating data from those applications:

1. A new event within the third party event generator is passed on to the IOIDS using a local procedure call, which results in direct processing of this information.
2. All event information from any third party event generator is mirrored into a central database, where the IOIDS may pick up the events using trigger technology and processes the information afterwards.

Both approaches require a normalisation of the data on the third party application side. With SoapSy (see section 6.2.3 for details) a database has been developed, which is able to store event information from very different applications in a normalised way by avoiding loss of information due to employment of application specific extensions. The integration of event information from third party applications is implemented using the SoapSy approach in conjunction with trigger mechanisms.

Responsibilities

From the collection of objectives the following list may be applied to the integration facilities for third party event generators:

- Translation between application specific data format and the IOIDS common data structures
- Notification in case of new events

Implementation

The implementation for the third party integration mechanisms comes in two parts, namely:

1. An output plug-in (or similar approach) for the third party application, which enables this one to transfer event data into the normalised IOIDS data repository.
2. A module for each application inside IOIDS, which enables IOIDS to understand the extension information for the third party events.

The output plugins may be implemented without any connection to IOIDS, which appears to be another advantage of the approach. Some information about these plugins are discussed in the experiments and analysis part in section 7.3.

The implementation of modules for each application to be understood within IOIDS are part of the project. Without such modules, IOIDS could only process the common core information by leaving the application specific extension information behind. The integration of such IOIDS extension plugins is realised using modules with the following attributes:

- New events from any third party event generator are realised by IOIDS using trigger mechanisms.
- For each third party application there must exist one extension module.
- The plugins support fetching of application specific information as well as their storage.

- The corresponding plug-in is loaded dynamically determined by the event type given in the core event.
- Due to the generic approach of the datatypes within IOIDS and their close relation with the database design extension plugins may be implemented easily for supporting the mapping.

The design and implementation of this integration mechanism is discussed as part of the IOIDS design and realisation. Conceptual issues are addressed in section 6.2.3 whereby attention to implementation of the integration is drawn in section 6.3.2.

4.5. Analysis and Evaluation

The analysis and evaluation of the Inter-Organisational Intrusion Detection System approach as presented appears to be difficult due to the lack of similar approaches in the area. Consequently, only several sub sets of the features of IOIDS may be mirrored against available approaches and after all a picture may be drawn, which represents the overall feature set.

The evaluation of IOIDS is carried out in the following two steps:

1. Analysing of features of similar or related approaches by interpretation of publications and documentations about such ones. This way the supported features of such approaches and applications may be mirrored against the ones of IOIDS. For this step of the evaluation process the entire list of available products to date from section 4.3.1 is integrated using the attributes listed and explained under section 4.3.2.
2. A practical evaluation in a laboratory environment shall examine, what results IOIDS is able to achieve in practise. For this step only a subset of the available approaches from section 4.3.1 will be used. Although IOIDS has not been developed with major focus on performance part of this step will draw attention to achievable results in the context of benchmarking.

The requirements and idea in detail for each of the two steps is explained in the following two sections.

4.5.1. Comparison of approaches

Loads of research is going on in the area of network security and intrusion detection in particular. Not all efforts are resulting in an application or some of them have not yet reached this status. However, publications and documentations draw a picture of the approach in

question, which provides enough information to perform a comparison based on features as they are described.

Execution

A list of low-level features, which are thought to be important for a scalable distributed intrusion detection system infrastructure has been presented in section 4.3.2. This list will be applied to a matrix; each of the approaches introduced in section 4.3.1 will appear with a column in the given matrix. By examining literature about the approach in question such as publications, documentations or published test results the corresponding value will be applied in the given cell of the matrix. The following values may appear inside the matrix:

Y – Yes, this feature is supported.

N – No, this feature is not supported.

N.A. – Not Applicable; the feature in question cannot be applied to the approach.

In the case of a requirement of further explanations for a certain value, footnotes will be applied in order to eliminate uncertainess. The option for *Not Applicable* is required due to the aforementioned effect that certain projects are only supposed to cover a certain subset of features. Attributes outside this set will be marked as *N.A.*.

The execution of the literature based analysis of IOIDS is provided in section 7.2 of the experiment and analysis chapter.

4.5.2. Experiment

The practical part of the analysis shall show the applicability of IOIDS to real networks and shall examine, whether the implementation holds up with the features described for IOIDS in the theoretical analysis.

Due to the different components of the overall architecture it is reasonable to execute the experiment in several stages; each analysing a different set of modules. The three stages for the experiment of IOIDS are:

1. Infrastructure experiment (G4DS)
2. IDS event exchange evaluation (IOIDS)
3. Integration with data engine

The setup and execution plan for each of them is discussed in the following sections.

For the execution of the experiment a laboratory environment has to be set up. Figure 4.2 pictures the proposed topology in its simple structure to be used for stage one of the experiment.

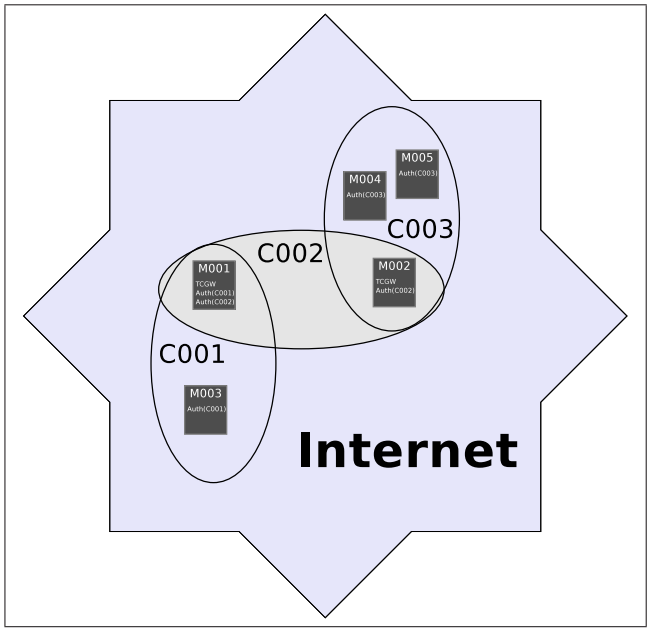


Figure 4.2.: Proposed network topology for experiment - stage I

In detail, the experiment topology is marked by the following attributes:

- Three communities are present in the topology; their identifiers are *C001*, *C002* and *C003*.
- Five members are present; each of them responsible for different tasks. In detail these ones are:

  1. Member *M001* is member of the communities C001 and C002. For both of them it as acting as a community authority. Furthermore, it should act as a gateway between the two in order to pass messages on from one to another.
  2. Member *M002* is member of the communities C002 and C003. For community C002 it holds additionally the authority role. M002 is the gateway between the two mentioned communities.
  3. Member *M003* is only present in community C001. It employs the role of an authority for this community.

4. Members *M004* and *M005* are only present in community C003. Both them are acting as authorities for this community.

- Low level communication is provided by a TCP/IP based network as used in the Internet.

The given network infrastructure is a fully connected G4DS network; meaning that every node is able to communicate with any other node from the topology. Routing on application layer through the network should be provided by the two gateways M001 and M002.

For stages two and three certain additions have to be applied to the topology. Intrusion Detection components have to be deployed at certain locations as well as receivers for knowledge have to be determined (receivers will be reading from the network only; they are not supposed to contribute knowledge).

Stage I - G4DS

In *Stage I* the subjacent communication infrastructure *Grid for Digital Security* (G4DS) will be evaluated. Messages will be traced throughout the network in order to check whether population and routing are working properly. A small and light-weight sample application examines the attributes of G4DS without causing too much overhead on the knowledge grid application layer. Last but not least G4DS‘ protection and availability mechanisms will be examined using penetration methodologies. In detail, the experiment shall be carried out the following way:

Sample application A sample knowledge service in form of a simple chat application making use of the G4DS communication platform examines features of G4DS by avoiding big overheads due to its simplicity. The sample application is marked by the follwing features:

- The new service is valid for all available communities C001, C002 and C003.
- Any node may send a message at any time.
- Distribution domains of messages may range from single users through all members of a community up to all known members of the services on the local node.
- Plain text messages will be used without any parameters. (This has to be stated in the service description)
- Only nodes, which are currently connected to the service will receive the messages. Otherwise data will be dropped - due to reasons of simplicity there is no need for the recovery of such messages.

A service description has to be developed for this knowledge service and has to be applied to all nodes of the infrastructure. At least two nodes have to be chosen for acting as service authorities for the application.

Messages shall be exchanged from all nodes to all nodes within the topology. The execution must consider the following requirements:

1. Messages must be delivered from each G4DS node to each other. (as long as the receiver has the service in question started up and is connected to G4DS)

2. G4DS access control must apply rules on certain nodes, which prevent messages from being passed on to the connected chat application. (this way, messages will still reach the G4DS, as required in item 1, but they will not reach the given application)

3. Data of wrong type shall be attempted to be sent to a node (XML data or binary encoded data).

After carrying out the given steps in an initial step, they will be re-invoked again and again under penetration circumstances.

Penetrate The penetration test of the G4DS network shall cover the following attack patterns:

Sniffing – Data shall be intercepted at a node within the network which is not the final receiver of the message. Sniffing shall be performed on both layers:

- *G4DS Layer* – A router or so-called Trusting Community Gateway (TCGW) shall intercept messages and try to recover data carried inside them. This must be carried out by implementing extensions to the G4DS implementation.
- *TCP/IP Layer* – A generic network sniffer such as Ethereal or Tcpdump shall be placed at a location inside the network, which enables listening to traffic, which the node is not the destination for. By examining parameters and raw data within the several TCP/IP protocol stack layers it shall be attempted to draw conclusions from the content transmitted.

DDOS - (Distributed) denial of service – By the use of (distributed) denial of service attacks such as flooding and the like it will be attempted to prevent a certain node or a certain set of nodes from continuing their work. Again, these attacks will be carried out on two layers:

- *G4DS Layer* – On G4DS layer it is attempted to break down the G4DS service on the node in question. This may either be achieved by flooding the victim with G4DS messages (likely in large sizes in order to request loads of processing time on that service) or by introducing DOS vulnerabilities into the G4DS code, which enables easy shutdown of the node in question. Both approaches will require code modifications for G4DS.
- *TCP/IP and OS Layer* – For this option it is not attempted to attack the G4DS service on the node in question but the operating system or other resources on the machine itself. For these reasons well-known attack patterns for the operating system may be used, which will prevent it from working; or the node is flooded with large network units in order to prevent it from or slow down the processing of network traffic.

For these attack patterns it is not of most interest how much the single node is resistant against DDOS attacks, but much more in which conditions the overall infrastructure is able to maintain its processing and distribution of information.

Spoofing - With this attack pattern a node shall attempt to insert information into the grid system by pretending to be a different node. This task will be carried out on both layers again; firstly by attempts of spoofing G4DS member information and, secondly, by attempts of spoofing attacks on TCP/IP layer such as IP spoofing.

It shall be examined, whether G4DS is able to work out that the traffic has not been initiated at the node it claims to be sent from.

Stage II - IOIDS

Stage two of the practical evaluation process deals with the core functionality of the IOIDS knowledge service. The following issues shall be addressed in this stage:

- Is the communication within IOIDS using the Grid for Digital Security platform persistent and performant enough? It has to be examined, whether the communication infrastructure supports required distribution domains appropriately, meaning that messages are delivered either to single destinations or to a certain set of destination nodes (such as all members of a community) in reliable and performant way.
- The persistence of the IOIDS system shall be evaluated; meaning whether the database type as well as the database model are adequate for storing the processed information. Additional attention has to be drawn to the possibility of storing communication

platform dependent parameters such as the sender of a message or the corresponding community, the knowledge chunk in question was travelling in.

- The access control mechanisms provided by G4DS may be utilised in order to prefilter content for the IOIDS service. Access control policies should be implemented and their triggering within G4DS must prove their applicability.

For execution of stage two of the experiment the IOIDS implementation as discussed in chapter 6 will be deployed to the laboratory environment as pictured in figure 4.2. The following changes have to be applied to the existent infrastructure from phase one:

- The IOIDS application has to be deployed on every node of the G4DS network.
- As part of this process a database for persistence of audit data has to be deployed for each of the IOIDS nodes. They may be installed on the same node as the IOIDS service is running on.
- A service description for the IOIDS application is created as part of its implementation. This description has to be deployed to the G4DS system on each of the nodes, where IOIDS has been installed on.
- IOIDS access control rules have to be developed and merged with the rule set already deployed for the G4DS system.
- The dataengine is configured in a way that every incoming local event is distributed to all known members of the service. Finer granularity on the rules will be provided in the third stage.

After applying the aforementioned changes to the laboratory environment the following steps have to be executed and recorded in cycles again and again in order to cover the use cases discussed afterwards:

1. Make sure, the network is isolated in a way that no unrequested data may tamper with the test results.
2. Check and record the status of the event databases on the nodes of the infrastructure.
3. Check and eventually adjust access control rules on the nodes of the infrastructure.
4. Make sure IOIDS services are running on all nodes and connected successfully to database backend and the G4DS communication platform.

5. Manually insert events into the event database which should be triggered and picked up by the IOIDS system afterwards.

6. Check and record the status of the event databases on the nodes of the infrastructure and evaluated changes, which have occurred since the check before the test.

7. Check event logs of both the G4DS system and the IOIDS system and evaluate the information in order to work out, how many and which messages have been passed on, have been sent or have been received.

The events for the experiment in this stage will be created manually in form of XML documents to be sent to the database, which allows full control over their content. Changes to those events and the access control rules for the G4DS subsystem have to be applied in order to cover the following use cases (use cases have to be combined with each other):

- An event triggered from the database distributed to all nodes of the network.
- An event is sent off to a distribution domain but not accepted by a few destinations (access control).
- A single event with no related events.
- An event, which carries further related events with detail information.
- Several grades of details should be provided in the exchanged event data (meaning some of them contain information about attacker, victim, etc others in contrast do not).
- The same event is sent and received several times. Both events are identical including timestamp information.
- The same event is sent and received several times. Both events are identical apart from their timestamp information.

Besides the examination of messages being delivered as they are supposed to, measures should be taken to give answers to performance issues. These include:

- How many events may be processed within a certain time frame?
- How much time does it take between an event occurring on the first node and being integrated in the data pool of the last node.
- How much data is carried within the messages. Measures have to be taken for:

  - The raw event data
  - The IOIDS event message
  - The G4DS message

  This will allow us to make a statement about the data overhead caused in each layer.

Stage III - Data engine

In the last stage of the practical IOIDS evaluation process the complete feature set of the overall system shall be examined. This way attention will be drawn to the following issues:

- The audit data repository with all its features for triggering and picking up new events including their third-party extension information.
- Full functionality of the IOIDS dataengine which includes applying of different values for protection of new event information as well as population of audit data to different sets of destinations.
- Installation of members within the network which will pass event information received within one community into another community.
- Full integration and evaluation of access control mechanisms including the support for classifications of knowledge chunks.

In order to execute the stage III experiments further extension have to be applied to the laboratory environment (see figure 4.2) on top of the changes from stage II. In detail these extension are:

- Installation of third-party event generators, which will contribute real world audit data for the laboratory environment.
- Implementation, integration and deployment of IOIDS modules for the third-party applications in question in order to process their application specific information.
- Development and deployment of IOIDS dataengine policies, which define detailed reactions to certain events occurring on the node.
- Development of access control rules in order to support the filtering of messages based on message classifications and their integration with existent G4DS access control mechanisms.

- Installation of invader stations and applications within the laboratory network, whose appliance will trigger certain events on the third-party applications.

Furthermore, for reasons of comparing features and shortcomings of IOIDS with the ones of existent technologies, those ones have to be deployed and configured as well. For the practical analysis the following products will be installed for this purpose:

- Prelude
- SnortNet

Both them have been discussed in more detail in section 4.3.1 in the overview of representatives. They have been chosen for laboratory tests due to their popularity and the availability of implementations for them. They have to be deployed and configured, event databases have to be set up for them.

It is essential, that all tests are driven under exactly the same conditions for each of the three approaches. In order to avoid interference between the approaches, however, they should not be running all together at the same time. Consequently, before each test run, the laboratory environment has to be brought into a consistent and well-documented state. One of the three approaches will be deployed and the test will be executed; afterwards the state of the environment has to be rolled-back for the next approach in order to guarantee identical conditions.

In detail, the following steps should be executed for each use case (which will be described afterwards):

1. Choose the first approach for executing the test.
2. Make sure, the network is isolated in a way that no unrequested data may tamper the test results.
3. Bring the laboratory into a well-defined state and record the status.
4. Check and record the status of the event databases on the nodes of the infrastructure.
5. Check and eventually adjust access control rules on the nodes of the infrastructure (if supported for the approach in question).
6. Make sure all required services are running on all nodes.
7. If required start-up third party event generators.

8. Penetrate the network with the prepared invader tools.
9. Check and record the status of the event databases on the nodes of the infrastructure and evaluated changes, which have occur ed since the check before the test.
10. If available check logging information of the approach in question.
11. Roll-back the entire laboratory environment to the state described in item *4* and protocol the status.
12. If more approaches are available, deploy the next approach and continue the test with item *5*.

Afterwards, results have to be processed and normalised in a way that enables us to compare between the three approaches.

The use cases for stage III should cover the following conditions:

- The infrastructure is fully functional and shall handle a real-world like amount of data.
- The network is penetrated significantly in order to generate a high volume of audit data.
- Certain nodes within the infrastructure are not or not fully functional. This may either be achieved by DOS attacks or simply disconnecting the nodes in question from the network.
- All nodes within the infrastructure have established equal trust relationships between each other.
- Nodes within the network want to distribute different levels of information to different distribution domains.
- Nodes within the network want to process information from different sources differently, meaning to accept and store knowledge from group *a* but dropping information from a certain group *b*.

Benchmarking

The benchmarking part of the experiment shall examine certain performance features of the IOIDS system in comparison to similar approaches as described in stage three of the practical evaluation (4.5.2). IOIDS only provides a proof-of-concept implementation and, this way, has not been developed with major attention on performance issues. This way, the benchmarking

part does not make up a big part of the experiment and is rather been executed as part of stage three. It is rather a structured overview of results gained from this stage.

Using those results, the following information should be compared for the three approaches:

- How many events may be exchanged within a certain time frame?
- How much time passes from the detection of an event on one node and the integration of this information on the destinations?
- How much network traffic is produced by the approach in question?

4.6. Conclusion

With the given information in this chapter an overview has been laid out of the architecture and implementation for the proposed Inter-Organisational Intrusion Detection System infrastructure. Proposed modules have been separated and introduced including their references for detailed information about each of them. Last but not least, a structured methodology has been described, which will be used in chapter 7 for carrying out the analysis and experiments.

In the upcoming two chapters the involved modules will be explained in detail. Firstly, the subjacent communication platform Grid for Digital Security is discussed in chapter 5. Afterwards, the actual Inter-Organisational Intrusion Detection System application, running on top of G4DS, is discussed in detail in chapter 6. Integration mechanisms for third-party event generators are discussed as part of this chapter in section 6.3.2.

Following these two chapters the analysis part (chapter 7) will draw attention to the analysis process itself and the execution of the experiments. Features of IOIDS will be examined and compared to the ones of similar or related approaches as described in this chapter.

Chapter 5.

Grid For Digital Security (G4DS)

5.1. Introduction

This chapter provides detailed information about the way, Grid for Digital Security (G4DS) is operating. Aligned to the development process the remarks here are broken down into architecture and implementation. The Architecture bit is providing kind of a general overview of the G4DS approach, and this way visualises, how components interact and how the whole system is hold together.

The Implementation section in contrast draws attention to the approach in more detail and this way provides information about employed implementation patterns; finally it shows how the complexity of G4DS could be broken down into pieces and how each of the modules was realised.

5.2. Architecture

As already outlined in the objectives (section 3.3.1), modularity and re-usability have been major issues for the design of G4DS. A communication platform was to be put into place, which implements a secure and reliable information exchange platform and provides interfaces for its use to applications, the so-called services.

No matter what kind of data the connected service is willing to exchange, G4DS is able to transport it. Consequently, G4DS is able to carry all of the following types of data:

- Unstructured / unknown text data
- Structured text data such as XML or the like
- Binary data

This could only be achieved by employment of strict separation of G4DS control information and payload data. G4DS itself does not care about the data it is carrying; due to encoding

and encapsulation mechanisms (see section 5.3.2), it is completely flexible from this point of view.

5.2.1. Overview

In order to understand the mode of operation of G4DS, Figure 5.1 provides an overview, which visualises G4DS as a black box. Major attention has been drawn to Configurability of G4DS; hence, to enable each node to configure its G4DS instance for its needs.

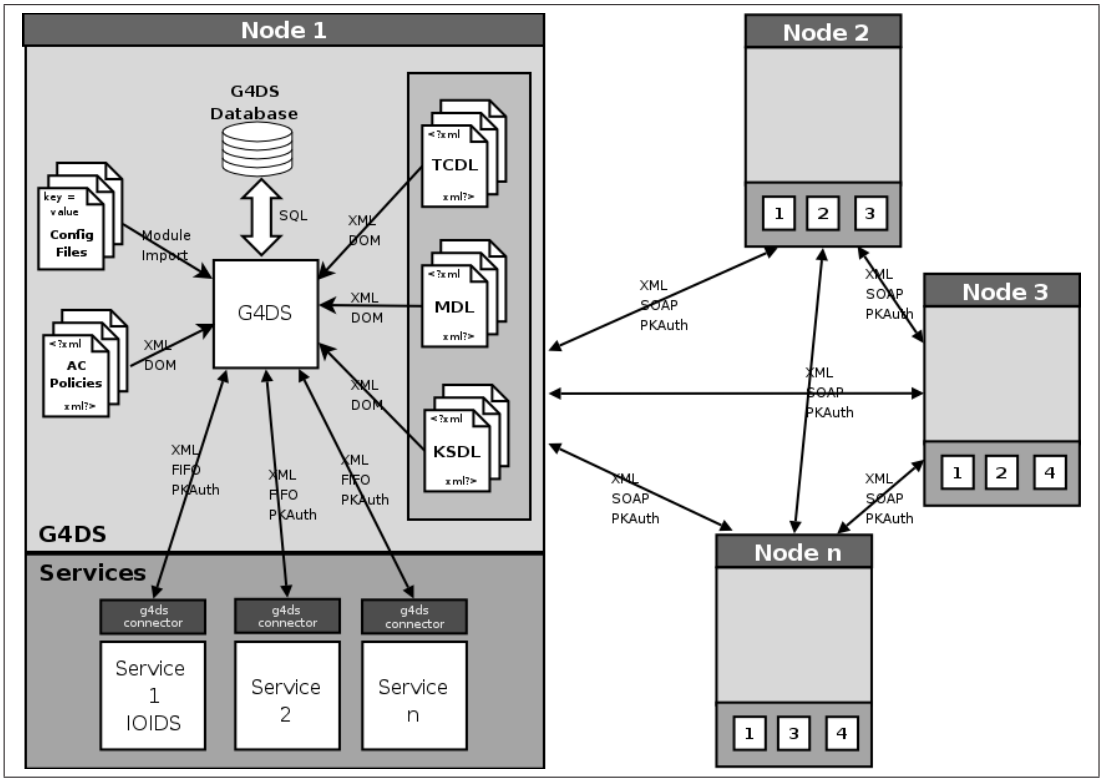


Figure 5.1.: G4DS Base Layout

The following documents are processed by the G4DS system:

- *Configuration Files*: The starting point for all input is configuration files, which tell the G4DS core about the required descriptions to load up into the system at booting time. (see section 5.3.3 for details on configuring and setting up G4DS)
- *Member Descriptions* (MDL), *Community Descriptions* (TCDL) and *Service Descriptions* (KSDL): The objects, G4DS is dealing with are either of the type member, community or service. For each instance of them an XML-based description must be created

which is then processed by the G4DS description processor. Detailed information is provided for the descriptions in section 5.2.3 and their processing in section 5.3.3.

- *Policy Files*: An access control mechanism has been implemented for G4DS. It bases its decisions on knowledge from local managers and the policy files. The XML based policy files are processed and its information is transformed into ordered rules for an access matrix. More information about access control is provided in section 5.3.4.

For reasons of persistence the G4DS system is connected to a database, in which all descriptions are stored and relations are maintained. This database is totally unlinked from any database the connected application might employ for its needs. Finally, a number of different services may connect to the G4DS system in order to make use of its communication facilities.

Besides the configuration files all the aforementioned information and communication is XML encoded. This way it may be easily processed into a DOM (W3C) tree, which makes the given information easily accessible. As outlined in Figure 5.1, the services connect to G4DS using FIFOs[1], which are accessible via operating system path names. Security and integrity are ensured here by employment of private / public key authentication and encryption. (see also section 5.3.6)

Same way the connection between the peers is secured by employment of private / public key authentication and encryption. However, the connection between them is established using well-known network protocols, such as the XML based SOAP[2] protocol (Box et al. (2000)) or generic TCP/IP[3] sockets (Stevens (1994)). In fact, other transport protocols may be integrated with G4DS easily using a plug-in mechanism. (see section 5.3.9 for details.)

5.2.2. Database layout

Each node has to store its on view of the G4DS topology. In order to make all the different entities with their relations between each other persistent, a database as media is most appropriate. Consequently, a database layout has been developed, which is able to picture the entire network structure from the view of a certain member. Figure 5.2 visualises the db layout; afterwards the components are explained in details.

The central entities in the given layout are *Communities*, *Members* and *Services*. Class Communities maintains all information about Trusting Communities, Members about members and Services about applications or so-called knowledge services. The remaining entities

[1] Fist in First out - information access approach, where messages are processed in order of their arrival
[2] Simple Object Access Protocol
[3] Transmission Control Protocol / Internet Protocol - commonly used protocol for establishing session based network connections on layer 4 of the TCP/IP protocol stack

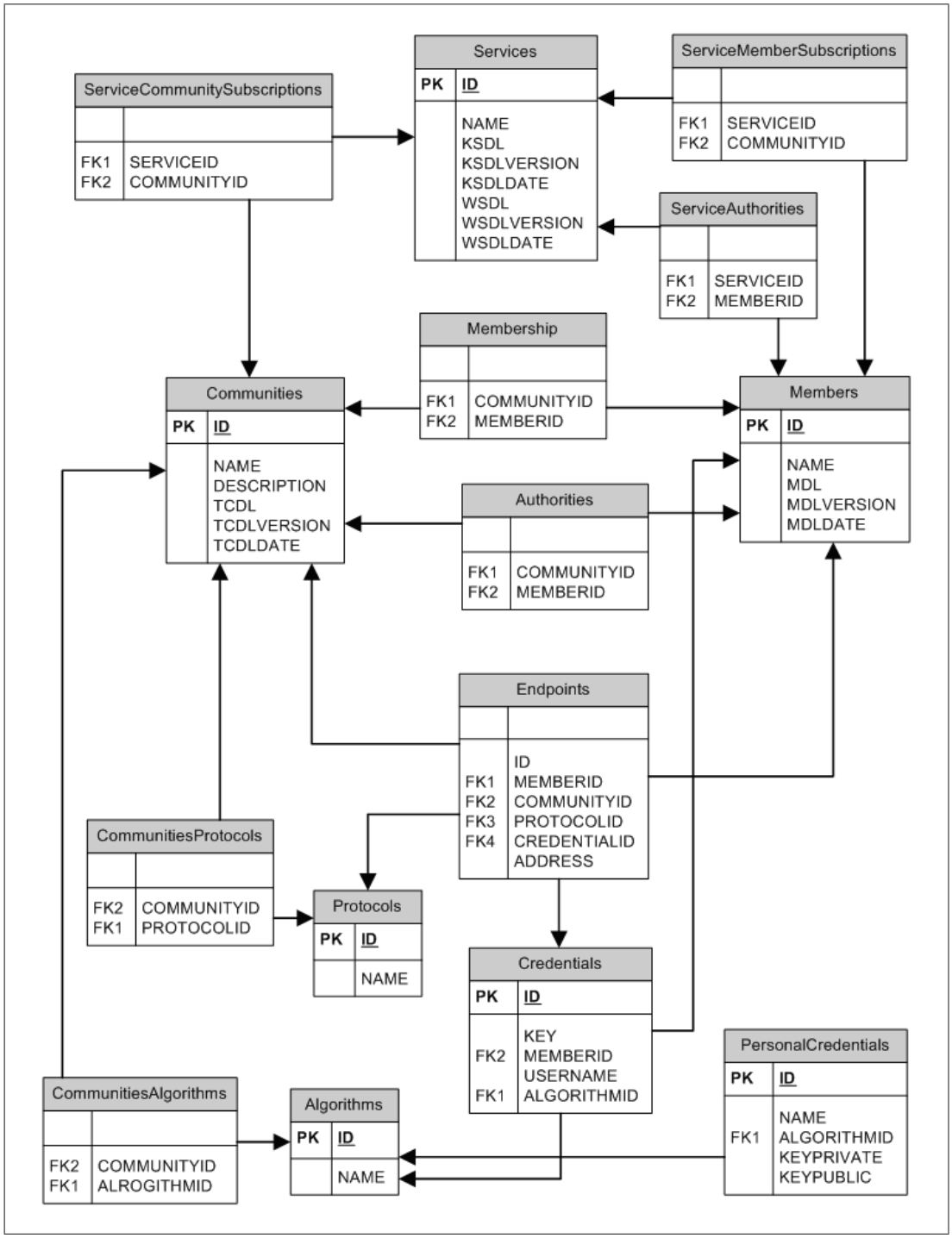


Figure 5.2.: G4DS Database Layout

are supposed to support with the relations between the three major tables or add some additional information.

Membership and Authorities

The most important role for communities and members as well as for services and members is the one of membership. Since these relations are always n:m relations (e.g. each member may be member of different communities as well as each community has several members belonging to it) an additional table had to be introduced for each relation. This way, the following tables are in place:

CommunitiesMembers – Maps the membership of members (nodes) to communities.

ServicesMembers – Maps the direct affiliation of members to services.

ServicesCommuntities – Instead of subscribing each member of a community to a service, the entire community might be subscribed to it. This table is responsible to map this behaviour.

Additionally, the tables *CommunitiesAuthorities* and *ServicesAuthorities* have been introduced. They maintain the special role of a member to be an authority either for a community or for a service.

Algorithms and Credentials

Since a public key infrastructure (PKI) has been employed for the G4DS network (see section 5.3.2 for details of securing the communication channel between G4DS nodes), there is a need to store public keys of remote members in order to authenticate them as well as validate the integrity of messages. However, public keys do not come as a isolated piece of information, there are often accompanied by a username. Furthermore, they have been created using a certain encryption algorithm and are only valid for this one. All these pieces of information (public key, user name and algorithm) are integrated in a table called *Credentials*. The name of the algorithm indeed is not stored simply as a string but as a reference to an Algorithms table. This way, it is ensured, that names for algorithms are normalised.

Another issue is arising when focussing on authentication and related security measures, namely the credentials on the node itself. Same way, they have to be connected with a certain algorithm. Additionally to the public key, as maintained for remote nodes, a private key has to be stored. These pieces of information are integrated in the table called *PersonalCredentials*. Again, the algorithm is identified by a reference to the Algorithms table. There is no need

to reference any member since these credentials are intended to be used as properties of the local node only.

In the descriptions for trusting communities it was pointed out, that a policy has to be brought into being, defining how to communicate; meaning, which protocols are utilised and which algorithms may be used within the community. Exactly this behaviour is mirrored with the table *CommunityAlgorithms* for the relation between communities and algorithms. It defines, which algorithms are utilised within the community; hence which algorithms have to be supported by each node, which is member of this community, and which each node has to provide credentials for.

Protocols and Endpoints

Many different protocols (such as SOAP, HTTP, SSH) are to be used for G4DS. In the *Protocols* relation the local node stores, which protocols it is aware of. This way, names of protocols are normalised and other tables do not store the name of a protocol directly, but reference the corresponding entry in the Protocols table.

The relation *Endpoint* finally puts all the information into relation. In fact, it provides some kind of address resolution for G4DS system. A simple one-to-one resolution for addresses, as employed for the Internet naming system DNS (Postel (1994)) may not be utilised due to the different protocols communities and nodes may employ. E.g. an address for a SOAP connection is made up by a URL, a SSH connection, however, will simply point to a certain IP address or DNS name and a TCP port. The table Endpoint is able to deal with the problem of different protocols, and additionally, integrates the information about credentials to be used for the certain node and certain protocol. Finally, one entry in the Endpoints table gives information, how a certain node through a certain community using a certain protocol may be reached; hence, which address and which credentials are to be used for this constellation of parameters. (The employment of communication facilities themselves are explained in detail in section 5.3.2.)

Finally, a table called *CommunitiesProtocols* is maintained, which, as described above for the table CommunitiesAlgorithms, maps knowledge about community policies. In fact, it says, which protocols are allowed to be used for inside community communications; hence, which protocol has to be implemented by each of its members and this way, endpoints for which protocols must be provided by them.

Services

G4DS has been introduced to support applications by providing a reliable and secure communication infrastructure. Applications using G4DS are called services. G4DS must have certain knowledge about the connected services in order to carry out its role.

For these purpose a relation *Services* has been introduced which maintains all necessary information about services. The three additional table ServicesCommunities, ServicesMembers and ServicesAuthorities maintain the relations involving services in the corresponding manner as described for members and communities just before.

5.2.3. Descriptions

As outlined all entities within G4DS, namely members, communities and services, are described using XML syntax. Each description must be provided in a single file. In order to unify all descriptions an XML Schema[4] (W3C (2005)) has been developed for each dialect. The following sections discuss the description of each entity in detail.

Member Description

One member description has to provide all information related to a single member within G4DS. On the one hand, this is made up by general information about the member, such as:

- an unique member identifier
- a version for the description
- a name of the node
- the date of this version of the description
- some more information about organisation and geographical location of the node

On the other hand, information must be included in the description, which allows the member to be contacted by other members. These include:

- one credential (public key) for each algorithm, this node is supporting
- a list of communities, this member is subscribed to
- a list of endpoints for each community, telling other members of the community, how to contact this node

[4]Defines the vocabulary for XML documents. (Alternative to DTD)

All this information together is enclosed in member description message (MDL) tags, which defines it to be a member description. A sample member description and the corresponding XML schema are provided on the resources CD.

Community Description

Respectively, one community description holds all information related to a single Trusting Community. Again, the kind of information in there may be divided into general information and information supporting the operability of G4DS. The part of general information is made up by:

- a unique community identifier
- a version for the description
- a name of the community
- the date of this version of the description
- some more information about organisation and geographical location of the node

The operational information in contrast include:

- the communication protocols to be used with this community; hence, which each member has to support in order to operate with the community
- the authentication and encryption algorithms, which are supported within the community
- general information about the authorities for this community and, to a certain extend, how they are to be contacted via G4DS
- information about so called community gateways; meaning, via which nodes knowledge may be passed into this community from other communities and vice verse

The information from the given lists is all put inside a community description XML tag, which identifies it as such a one. The sample description for a community and the corresponding XML schema are provided on the resources CD.

Service Description

Service descriptions are put in place to describe services and how they interact within G4DS. General information for services are:

- an unique service identifier
- a version for the description
- a name of the service
- the date of this version of the description
- information about real world contact opportunities for the service

The set of information on the operational level is made up by:

- ID, name and description for each supported message format for the service (this addresses the application level message formats, e.g. plain text messages for a simple chat or IOIDS for a service exchanging computer incident related material)
- general information about the authorities for this service

An example of a service description for a simple chat serviceand the corresponding XML schema are provided on the resources CD.

IOIDS as one service of G4DS discusses the idea of services from the view of the client application and, this way, provides more information about service descriptions in section 6.3.1.

5.2.4. Access Control

In the objectives (section 3.3.4) the requirement for a strong and reliable access control mechanism for G4DS has been discussed in detail. It must support with the protection of information in a simple and configurable manner whilst allowing population of knowledge to the highest possible grade.

The access control system of G4DS is based on rules and chaining of rules. Similar approaches have been implemented for packet filtering firewalls such as the Open Source packet filter Netfilter / IPTables. (Netfilter (2005))

In principal, three types of objects are present in the permission model:

1. Actors

2. Targets

3. Operations

Actors perform certain actions on certain targets. Mapped into the G4DS system, actors are always members. Targets might be either members, communities or services. Operations are represented by strings, which are organised in a hierarchy. An example for the hierarchies of action strings is given for the control messages for the member sub-control system:

- g4ds (describes all actions for G4DS)
- g4ds.control (describes all actions for the control system of G4DS)
- g4ds.control.member (describes all actions for member control subsystem of G4DS)
- g4ds.control.member.read (describes all actions for member control subsystem of G4DS, which require read access)
- g4ds.control.member.read.requestmdl (is the action string for requesting a member description)

Rules may either apply to an action in particular or to a set of actions using the group identifier of any level within the hierarchy. For each rule a rule identifier has to be defined. Corresponding to this key, the rules are put in order at processing time. As important the order is for packet filtering firewalls, it is for G4DS access control. Each action could hit several action strings from the provided policies; so, the first rule suiting the given action string, actor ID and target ID is applied.

The rules are defined in so-called policy files which must contain a well-defined XML based representation of rules. Their structure is explained in further detail in section 5.2.4 and their processing and application in the implementation of Access Control in section 5.3.4. After processing the rules at boot-up time of G4DS, they are kept in memory in form of an access matrix for easy and quick access.

Matrix and ordered list

After processing the policies, the rules are kept in memory by G4DS in structure of a matrix. Regarding to Figure 5.3 the X-Axis of the matrix represents the actor, the y-axis is occupied by the target. The value of x and y is an ordered list of couples, made up by an access control action string and an access control reaction.

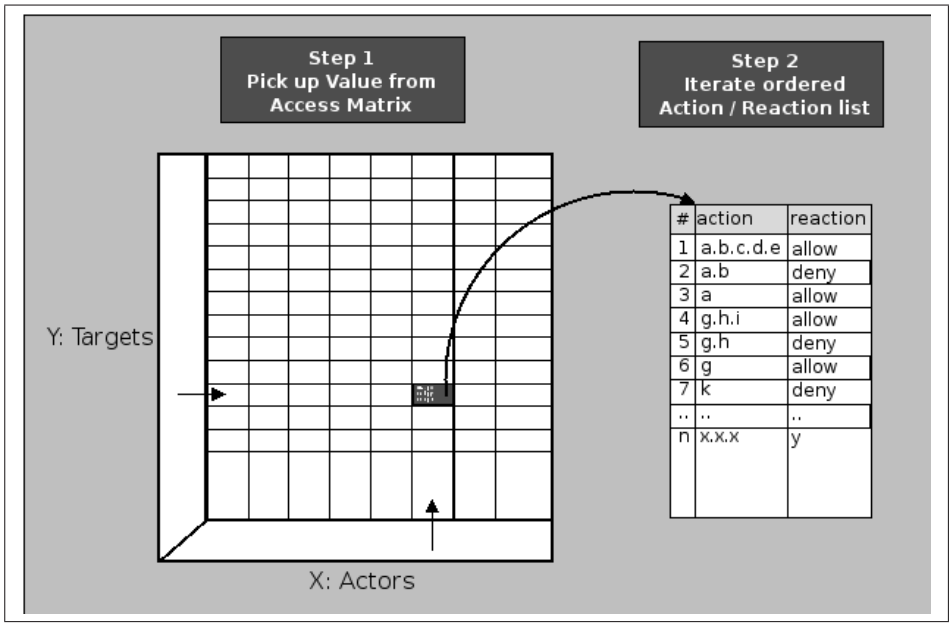


Figure 5.3.: Two stages access control

For validating any access using G4DS access control, all three parameters have to be provided, actor, target as well as action string. This way, the appropriate list within the matrix may be determined and is iterated, whereby each action value within the list is compared against the given action string. The corresponding reaction ID will be returned to the user of access control consequently.

Policies

As outlined in the access control introduction, the policies for its configuration have to be provided in XML format in so-called policy files. The policies have to be well formed; hence, aligned to the XML Schema for G4DS access control policies. The XML schema may be found on the resources CD. Recapitulatory, a set of policies needs to contain roles (which may be of any of the types, actor role, target role or operation role), the grouping for roles and finally the rules, putting all the information together. Rules are grouped in Rulesets. Listing 5.1 illustrates a simple example of policies for defining a group of trusted actors, which shall be allowed to any action on the local node.

Listing 5.1: Simple example for Access Control Policy

```
g4dspolicy
  roles
    roleset (* type = 'actors')
```

```
      role
         name = trusties
         description = Actors I would allow everything.
   roleset (* type='targets')
      role
         name = all_targets
         description = All targets

groups
   group
      rolename = trusties
         representatives
            representative (* type='member') = M001
            representative (* type='member') = M002
   group
      rolename = all_targets
         representatives
            representative (* type='membergroup') = *
            representative (* type='communitygroup') = *
            representative (* type='servicegroup') = *

rules
   ruleset
      id = RS000000
      name = Allow Trusties
      rule
         id = R00000
         comment = members defined in the trusties group are allowed every action in
              g4ds
         actor (* type='role') = trusties
         action (* type='operation_id') = g4ds
         target (* type='role') = all_targets
         reaction (* type='direct') = allow
```

Policies may be defined in several files. The order of their definition does not matter since the order of their processing is determined by their rule ids, rather than the order in the files.

To start-up with, G4DS is coming with a set of default policies, which may be found on the resources CD.

Policy redirects - chaining of rules The set of rules might quickly grow in complexity and it will be hard to structure and order them by their rule identifiers only. Consequently, the approach of rule sets has been implemented, where each rule must belong to one rule set. In order to continue processing in another rule, the reaction within one rule has to be defined as 'redirect'. For these reasons, two reaction types are in place for G4DS access control, namely 'direct' and 'redirect'. For direct reactions the only supported values are 'allow' or 'deny'. For redirected reactions however, the value must be the ID of a new rule set, whose

processing will be started whenever the rule is hit. After finishing the processing of the given ruleset (which may redirect reactions to further rule sets itself) the processing is terminated for the current rule set, since one rule met the pattern for actor, target and action string when running into the redirect rule. Further information about access control and particularly the its implementation are provided in section 5.3.4 inside the implementation for G4DS.

5.3. Implementation

This section covers implementation details for G4DS. This way, it visualises connectivity to database backends, draws attention to the message transfer and also provides information, how data is generated, transported and processed on several nodes.

5.3.1. Managers and Database Connectors

In section 5.2.2 G4DS Database Layout the tables used for making all objects of the G4DS infrastructure including their relations persistent have been introduced. This section will explain how this information is accessed and how the access to the database is encapsulated from the program core. Also have a look on Figure 5.4 for some more details.

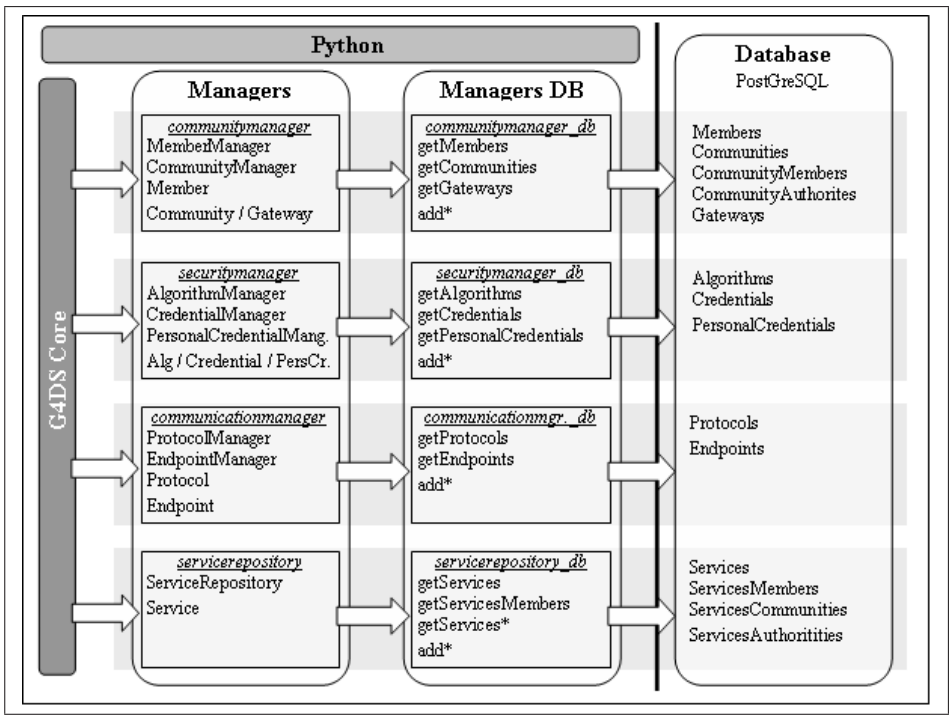


Figure 5.4.: Managers and their DB Connectivity

Managers

First of all, managers have been introduced, which are kind of containers for one special type of information. Managers have been grouped into the following categories:

- Managers for community and member purposes
- Managers for security purposes
- Managers for communication purposes
- Managers for service purposes

Regarding to the groups the manager are placed in different modules. The following four modules were introduced maintaining the given managers:

- Module *communitymanager*: Maintaining information about communities and members
  - Manager *CommunityManager*: Holding information about communities and their relations to members, authorities and gateways.
  - Manager *MemberManager*: Holding information about members and their relations to communities and gateways.
  - Additionally, there are classes Member, Community and Gateway, each of them representing one instance of the corresponding item.
- Module *securitymanager*: Maintaining information about algorithms and credentials.
  - Manager *AlgorithmManager*: Holding information about algorithms, in fact the relation between their names and identifiers.
  - Manager *CredentialManager*: Holding information about publicly available credentials; hence the public keys with corresponding information, for all known members.
  - Manager *PersonalCredentialManager*: Holding information about the private credentials of this node; hence the private keys with relations to the corresponding algorithm.
  - Additionally, there are classes Algorithm, Credential and PersonalCredential used for maintaining one instance of the corresponding item.
- Module *communicationmanager*: Maintaining information about protocols and endpoints.

  - Manager *ProtocolManager*: Holds information about protocols, in fact just about the relation between names and corresponding IDs for protocols.
  - Manager *EndpointManager*: Holds information about the endpoints. Endpoints, as already explained in the database section, are bringing all the bits of information together. This enables a user to just use an endpoint to assemble completely a G4DS message (incl. singing and encryption) and send it off.
  - Additionally, there are classes for Protocol and Endpoint. Again, they represent one instance for the corresponding item.

- Module *servicerepository*: Maintaining information about connected services. Exactly the same as the managers before, only because of the very different kind of data handled in there named as a repository.
  - Manager *ServiceRepository*: Holds information about services and their relations to members, communities and authorities.
  - An additional class for Service as one instance is provided too.

Managers DB Connectors

The managers themselves do not directly connect against the database. Instead, each of them maintains a reference to a database manager which is encapsulating the information of querying the database.

The same grouping as used for managers has been employed for the database connectors for managers. Consequently, the four modules communitymanager_db, securitymanager_db, communicatonmanager_db and servicemanager_db have been introduced. Instead of providing a class for each real manager the functionality is provided in functions. Figure 2: Managers and their DB connectivity does not mirror the complete collection of functions, instead it just provides an overview about the way of implementing it. This way, there are always functions provided for accessing the resources from the database, the so-called getters; as well as there are functions to append data for the database using the methods starting with add.

Another level of abstraction has been reached in the database connectors for managers by extracting the names of the tables to its own configuration module (not mentioned in Figure 5.4). This way, the database environment may be kept dynamic and table names may be aligned easily to new requirements.

5.3.2. Message transmission

XML has been constantly gaining popularity for message exchange. This is due to its simplicity, its extensibility, its all-round functionality and self-explaining grammar. XML was also chosen as the basic message format for G4DS.

Message Wrapping

Messages are organised in hierarchies. Each level of hierarchy should be separated; hence, should not need to know about any impacts concerning higher or lower levels of hierarchy. Consider a situation as pictured in XML representation in listing 5.2. First of all, each message to send via G4DS is a G4DS message. Then it is indicated, that the message inside the G4DS message is signed; this way it contains a section for the data itself and a corresponding signature. The raw data again belongs to a service, whose name is defined inside the service section.

Listing 5.2: XML Message Wrapping

```
<g4ds>
  <signed>
    <data>
      <service>
        <ioids> application specific data ? e.g. an event </ioids>
      </service>
    </data>
    <signature>abcd</signature>
  </signed>
<g4ds>
```

This simple example pictures already a requirement for wrapping again and again data into other higher level wrappers. However, the G4DS top level tag does not need to know, how a signed tag is assembled, instead it just needs to know where to pass the data, and this way, who knows how to disassemble it. In order to allow a wide range of data to be wrapped (including plain text or encrypted data) data is always wrapped using XML CDATA sections. This way, for each element the XML document may be parsed down to the element, which are relevant to this element; all wrapped data however will be encapsulated by a CDATA section and skipped by the parser. Instead, a string representation of it is returned, and the element handler itself should know what to do with it or to which location to pass it on.

This way, the example from Listing 5.2 looks in the CDATA wrapping version as given in Listing 5.3.

Listing 5.3: XML Message Wrapping using CDATA sections

```
<g4ds>
  <![CDATA[<signed>
    <data>
      <![CDATA[<service>
        <ioids> application specific data ? e.g. an event </ioids>
      </service>]]>
    </data>
    <signature>abcd</signature>
  </signed>]]>
<g4ds>
```

Note that the example in Listing 5.3 only explains the methodology, it does not exactly mirrors the messages as they are exchanged within the G4DS topology.

Nested CDATA sections One problem of the approach taken with wrapping using CDATA sections is that XML by its nature does not support nested CDATA sections. How should the parser know, where a CDATA section finishes? It may only rely on the first occurrence of a closing CDATA character sequence "]]>".

The way to get around this problem is the encoding of the data before putting it inside a CDATA section and decoding it after gaining the string, so that no further CDATA section beginning or end sequences are occurring inside the data. The way chosen for performing this encoding is hex encoding. This means that each character in the source data is mapped into its hex representation of its ASCII value; hence, mapped into two characters. This ensures that no forbidden sequences are left in the data and straight forward procedures for encoding and decoding could be implemented.

The problem from Listing 5.2 XML Message Wrapping using CDATA sections with hex encoded values is pictured again in Listing 5.4.

Listing 5.4: XML Message Wrapping using CDATA sections and hex encoding

```
<g4ds>
  <![CDATA[<signed>
    <data>
      <![CDATA[3c 73 65 72 76 69 63 65 3e 0d 0a 09 3c 69 6f 69 64 73 3e 20 61 70 70
           6c 69 63 61 74 69 6f 6e 20 73 70 65 63 69 66 69 63 20 64 61 74 61 20 96
          20 65 2e 67 2e 20 61 6e 20 65 76 65 6e 74 20 3c 2f 69 6f 69 64 73 3e 0d 0
          a 3c 2f 73 65 72 76 69 63 65 3e]]>
    </data>
    <signature>abcd</signature>
  </signed>]]>
<g4ds>
```

Note that only one level of hex encoding is shown in Listing 5.4. In fact, also the data from the tag <signed> on up to the corresponding closing tag </signed> would be hex encoded. This results in a hex encoding of the already hex encoded CDATA section inside the data element for the signed element.

The encoding on each level causes an increase of data by factor two. Consequently, the nesting of hex-encoded CDATA sections ends up in exponential growth. The way to tackle this drawback is the employment of compression facilities before the process of hex-encoding. Since we are dealing with XML encoded data, which is text data by nature, and this way may be compressed to a high rate, the employment of compression facilities cuts down the overall data exchanged via G4DS.

Implementation of message wrapping All wrapping issues are encapsulated from other modules in the module messagewrapper. Classes are provided in there, which are all derived from the super class MessageWrapper, which provide functions for wrapping and unwrapping for all required purposes. These are:

- wrap / unwrap G4DS messages
- wrap / unwrap control messages (more about this in section *Sending and Dispatching*)
- wrap / unwrap service messages
- assemble / parse XML encoded routing tables
- wrap for encryption / unwrap for decryption
- wrap for signing / unwrap for validation

The wrappers at this place perform some more actions then only putting the data into a CDATA section and wrap by an element of the given name. Certain methods need parameters, which have to be passed and encoded. For example, when wrapping an encrypted message, information about the utilised algorithm has to be stored together with cipher text, otherwise the receiver would not be able to decrypt the message on its side.

The measures for hex encoding are invoked automatically from inside these functions; hence, they are totally transparent to any requesting resource. Again, the functionality for hex encoding as well as hex decoding is encapsulated in their own functions, which are called from the aforementioned wrapper functions.

Encryption and Decryption

The theory for asymmetric encryption and decryption is that one encrypts a message at the sender's side with the public key of the receiver, so that only the receiver itself is able to decrypt the cipher text using its private key.

As outlined before a variety of algorithms for encryption and decryption may be used with G4DS. But how is outworked which algorithm and which keys shall be used for the requested action? Referring to *Protocols and Endpoints* (section 5.2.2) in the database section, all information about how and where to send a message is integrated in the endpoint. Whenever a message shall be sent, an endpoint has to be chosen. However, this is part of the section below named *Sending and Dispatching* and will be discussed there in detail. For now we assume that encryption and decryption are performed of the sending or receiving process and the endpoint is provided already whenever either of the two functions is invoked.

For a high level of abstraction a class SecurityController in its module securitycontroller was introduced. Functions in there are provided for encrypting and decrypting messages. In order to perform encrypting a name of an algorithm and a key (public key) together with the plain text has to be provided. For decrypting, however, only the algorithm and the cipher text is required - the personal key (private key) required for this action is loaded into the algorithm implementation at start-up time using the relations in the PersonalCredentialManager. The SecurityController loads the corresponding algorithm for the name from the AlgorithmController and passes the request for encrypting or decrypting to the algorithm implementation. This is possible due to the common interface all algorithm implementations have to provide when used for G4DS.

Signatures and validation The intention of signing messages and validating the signature with the message is to ensure that the message has not been altered on the way from the sender and, furthermore, to ensure that the message has really been originated at the sender as pretended. Common practise is to use the private key of the sender at the sender's side to sign the message. The outcome or so-called signature is transmitted together with the actual message and the receiver may use the message, the signature and the public key of the sender to validate the given message and origin.

From the implementation point of view this process is very similar to the one of encryption or decryption. Signing is performed as part of the sending process and validating as part of the receiving or dispatching process. The same class as used for encryption (SecurityController) is used here as well. Parameters have to be passed to the functions and the requests are thereupon passed on to the algorithm implementation, which may be loaded using the

AlgorithmController together with the name of the algorithm. All information required is integrated in the endpoint.

Sending and Dispatching

The sending and dispatching process brings all the aforementioned issues of wrapping, encrypting, signing and the like together. First of all, the central point for performing sending or dispatching is the module messagehandler with its classes *GlobalOutgoingMessageHandler* and GlobalDispatcher. The GlobalOutgoingMessageHandler may be accessed directly to send a message. The GlobalDispatcher provides a function *dispatch* which should be called by any protocol implementation whenever a new message arrives.

Sending a message There are two different kinds of messages to be sent via the G4DS infrastructure; namely service messages and control message. Service messages are messages from any connected application and are to be delivered to the same application on the receiver's side. Control message in contrast are G4DS internal messages and keep the system working and information up-to-date all over the network.

Consequently, two functions are provided in the GlobalOutgoingMessageHandler:

- sendServiceMessage: The G4DS wide unique ID of the service invoking the function must be provided together with an endpoint and the actual message to be sent. There is no limitation for the format of the message; it might be plain text, XML encoded data or data of any other type.

- sendControlMessage: Control messages belong to a control sub system. The unique id for this subsystem has to be provided together with an endpoint and the actual message to be sent. Since these messages are G4DS internal messages, they are definitely XML encoded; their detailed structure, however, depends on the control subsystem and does finally not influence its delivery.

Either of the two functions will call its wrapper function in the MessageWrapper and thereupon pass the wrapped message on to the final function for delivering G4DS messages, the *sendG4dsMessage* inside the GlobalOutgoingMessageHandler.

The function sendG4dsMessage will load all required information about algorithms, credentials and the like from the managers and perform the actions for signing the message and encrypting the message. Each of them is followed by a call in the MessageWrapper for encapsulating the output appropriately.

Finally, the result is wrapped in a G4DS message using the MessageWrapper and the protocol implementation as stated in the endpoint is loaded. The message is sent off using the sending function inside the protocol implementation.

Receiving and dispatching a message As stated in the introduction, the first locations for incoming messages are the listeners of the protocol implementations. These protocol implementations, however, are in charge to pass the message directly after receiving (and removing protocol specific header information) to the function *dispatch* inside the GlobalDispatcher.

This function will perform the inverse actions to the sender in the opposite order. This way, the incoming message is first of all passed to the MessageWrapper to unwrap it from the G4DS tags. Afterwards it is checked for encryption tags, if available, the algorithm name and cipher text is extracted using the MessageWrapper and the decryption is performed using the private key for this algorithm. Furthermore, the message is validated - by first of all extracting the information using the MessageWrapper, then recovering the endpoint of the sender and finally performing the validation. Afterwards, the nature of the message is determined; either being a service message or a control message. Either of them will be unwrapped using the MessageWrapper. Afterwards they are passed on depending on the nature; control messages are passed to the ControlMessageDispatcher inside the module g4dsconfigurationcontroller, service messages in contrast are passed to the ServiceIntegrator inside the module serviceintegrator. Messages are processed further in these classes. For more information check Controlling G4DS (Section 5.3.3) for control messages and G4DS and Communities (5.3.5) for service messages.

Message contexts Messages are wrapped several times before sent off and unwrapped after receiving. For example a MemberControllerMessage goes through the following wrappings:

- G4DS Configuration Controller Member Subsystem wraps it into a Subsystem Member message
- G4DS Configuration Controller itself wraps it into a G4DS Controller message
- G4DS Message Handler wraps it into a G4DS (plain) Message
- This message is encrypted and wrapped again into a G4DS (ciphered) message

At the receivers side each layer has to be unwrapped correspondingly. In each layer several kinds of information are transmitted. It would not make sense to unwrap the information and pass it on and on as parameters to the dispatcher for the next layer. Consequently, a

message context mechanism has been employed. Inside the module messagehandler there is a class MessageContextController which is able to maintain all the corresponding information for one message, identified by its message id. Each layer adds its information to the context and only passes the message id on to the dispatcher for the next layer. Finally, the processor of the message may access all required information (using the keys for the dictionary) and dispose the context for this message (id).

5.3.3. Controlling G4DS

A central module has been introduced for controlling all matters for G4DS: namely the *g4dsconfiguationcontroller*. Basically, this module provides two classes as its interface to the outside environment; the ControlMessageDispatcher and the OutgoingControlMessageHandler.

The former is used, as stated previously, by the GlobalDispatcher for passing on any ControlMessage. The ControlMessageDispatcher thereupon extracts the information from the received message using the ControlMessageWrapper within the module messagewrapper. As part of this the id for the requested control subsystem (MemberController, CommunityController, ServiceController or RoutingController) is returned. This will be used in order to pass the message on to the dispatcher of the corresponding Control subsystem.

The OutgoingControlMessageHandler in contrast is responsible for assembling control messages (invoking appropriate wrapping procedures) and pass it on to the global outgoing message handler. All messages created in any of the control subsystems should only be send through this interface, they should not connect to the global outgoing message handler themselves.

Each subsystem controller is in charge of controlling matters of a certain subject area. It has to provide a dispatch function in order to allow processing of incoming messages. In detail, the following control sub systems are in place:

MemberController

The member controller handles all requests concerning members. These include:

- Member description requests
- Replies to member description requests
- Member description update procedures

Member description requests request the latest description of either the receiver of the request or any other member of the G4DS infrastructure. These requests may be rejected regarding to the role of the requesting party in the permission model.

Member description update procedures are in place for providing facilities for each member to keep its own description up-to-date. Usually they will be pushed messages rather than polled messages (as for most of the remaining messages).

In the module messagewrapper is a class ControlMessageWrapper, which provides functions for the subsystem MemberController to wrap and unwrap its messages.

The dispatcher of the MemberController supports job continuation; hence regarding to the information in the job context about references it will not determine the requested action by evaluating the action flag itself, but instead it will put the extracted data into the message context (data) and thereupon the continuation of the scheduled job. (see section 5.3.7 for details on jobs)

CommunityController

The community controller handles all requests concerning communities. These include:

- Community Description requests
- Replies to Community Description Requests
- Community Membership requests
- Replies to Community Membership Requests
- Community Description Update procedures

Community Description requests are sent to a member of this community, usually an authority. They will then reply appropriately - means, whenever they have an up-to-date version of the description, it will be send. The community descriptions are not supposed to contain confidential data since they only describe, in which way the members communicate.

Community Membership requests are requests for a list of the known members of a certain community. Not for all communities this feature will be available (depends on the policy for the community). Replies will be generated regarding to the permissions of the member requesting.

Authorities are responsible for keeping the description of the community up-to-date. The development of a new version will usually involve manual intervention; the distribution, however, is performed by the system automatically. For these purposes functions grouped under the name Community Description Update procedures are in place.

As much as the MemberController the CommunityController is making use of wrapping functionality from the ControlMessageWrapper inside the module messagewrapper.

Additionally, it supports the job continuation; hence, control message for sub-system community controller with a reference id within the message are passed to the delayed job rather then processed directly. (see section 5.3.7 for details on jobs)

ServiceController

In order to control services, a service controller has been introduced. It is dealing with the following concerns:

- Request a service description for a certain service
- Reply to service description request
- Service description upload procedures (push mode)
- Request list of members, subscribed to a certain service
- Reply to member list request
- Request subscription to a certain service
- Reply to service subscription

Regarding to the community controller functionality, the procedures for downloading and uploading service descriptions is supported by the service controller. Again, service descriptions are not supposed to be confidential; hence, they may be exchanged throughout the G4DS topology.

One additional functionality for the service controller is the request of a list of members subscribed to a service. Practically, all authorities of a service maintain a complete list of all members subscribed to it. Whenever a member is interested in this list it may request it using this option. The reply, however, depends on the policies, employed for both the given service and the node, the request is sent to.

Finally, the service controller must support the automatic subscription to a service. In order to reply positively to this request, the receiver must be an authority of the requested service. Furthermore, the reply depends on the policy employed for the given service and the node, the request is sent to.

Messages for the service controller are correspondingly to member and community controller wrapped with the help of the ControlMessageWrapper.

Since replies are generated within the service controller, there must be a job on the other side to wait for a message; hence, to continue processing. Consequently, service controller can determine whether an incoming message is in reply to a previously sent request and will pass the message on to the appropriate job instead of processing it itself. (see section 5.3.7 for details on jobs)

RoutingController

Members of G4DS are grouped in so-called communities. Communities are not supposed to be isolated from each other; hence, there will be an overlapping of communities in regards to their memberships, and even more, members are supposed to pass on message from one community to another. This process is called routing. In order to make this inter-community communication working, nodes have to exchange information about gateways; hence, which nodes are allowed to pass on messages from which community into which destination community. All procedures concerning these matters are put together in the routing controller. The routing controllers provide the following functionalities:

- Download routing table (pull mechanism)
- Replies for routing table download
- Upload routing table (push mechanism)
- Replies for routing table updates

Each node is maintaining a routing table in order to know, which node it needs to contact first in order to get a message delivered finally to a certain community. (Check section 5.3.5 for details) For purposes of exchanging this information with other nodes, this routing table may be encoded in an XML message using functionality from the routing table manager and the routing table wrapper.

The processing of these requests depends on the local access control policy. Usually, routing information is not supposed to be handled as confidential; thus, routing table download requests should be replied with the encoded local routing table. Routing tables sent in push mode will usually only be accepted from very trustworthy nodes.

Since both approaches (pull as well as push mechanism) have to await messages replies, the routing controller also supports the continuation of jobs. Consequently, messages, which are sent as a reply to a previous outgoing routing message, are passed on to the corresponding job rather than processed by the routing controller itself.

Processing Descriptions

Several kinds of descriptions are being generated and processed all the time when working with G4DS. These include:

- MemberDescriptions (MDL)
- CommunityDescriptions(TCDL)
- ServiceDescriptions (KSDL)

In order to provide an easy way to generate and access the information all of them are XML formatted descriptions. (Their layout is discussed in detail in section 5.2.3.) An additional module named descriptionprocessor has been introduced to deal with all matters of generating and processing these descriptions. For each descriptions there is a class within this module; namely MemberDescriptionProcessor, CommunityDescriptionProcessor and ServiceDescriptionProcessor. The only action they perform is the parsing of the XML messages and the storing of the values in dictionaries.

However, more sophisticated actions may be carried out by these classes using their apply-functions. These ones are in charge to apply all the information stored in the dictionaries into the local managers (and this way into the database backend). In detail, they are performing the following actions:

- Process and apply the direct data for a member, community or service (such as ID, name or description)
- Process and apply referenced objects such as credentials (keys), protocols and endpoints
- Automatically, download, process and apply required descriptions for referenced objects

Solving Back-Referencing The last bit of automatic downloading and applying of descriptions comes along with problems due to references within the descriptions to each other. Consider the following situation: A new member description shall be applied. Member descriptions include information about subscriptions to communities of this member. This new subscription of the member now references a new community, not yet known on the local system; consequently this description has to be downloaded as well. If we further consider, that the member description in the first place describes a member who is an authority of the new community, the community description would reference back to this member and the description of this member would be requested for downloading again. The following procedure was implemented to get around this kind of problems:

1. Whenever a description shall be applied, it is first of all applied with its basic information excluding any referencing information.

2. Afterwards, the descriptions of the referenced objects (which are not yet known on this node) are download and applied. (Of course, each of them is running through the same process as well.)

3. The original description is applied down to all details and references.

This way, it can be made sure, that back referencing is not a problem anymore. However, the consecutive downloading of required descriptions is an expensive process. It should always run as a separate job (in a different thread) - check also Job context and job continuation. There is no termination for this process unless it's finished down to the last referenced object.

Maintain environment

In order to provide access to the configuration options for G4DS, a maintain environment is provided. Once this python program is started, it starts up a G4DS backend system and connects against it. The easiest way to describe its functionality is a screenshot of the menu as shown in Listing 5.5.

Listing 5.5: Interactive menu for G4DS maintain environment

```
                Maintain G4DS (Node M003)

Choose from the following options:

        General Options
        [s] ... Print status of local node
        [t] ... Send test message to node
        [l] ... Show latest log entries
        [y] ... Print permission matrix
        [z] ... Recalculate permission matrix

        Member Options
        [i] ... Print information about a G4DS node
        [m] ... Add / update member description from file
        [n] ... Export local member description to file
        [e] ... Add endpoints for member
        [p] ... Upload member description to remote nodes

        Community Options
        [j] ... Print information about a Community
        [c] ... Add / update community description from file
        [d] ... Add / update community description from remote node
        [u] ... Subscribe member to community

        Service Options
        [1] ... Print list of known services
        [2] ... Print infomation about one service
        [3] ... Add / update knowledge service description from file
        [4] ... Add / update knowledge service description from remote host
        [5] ... Push service description to remote host
        [6] ... Subscribe member to knowledge service
        [7] ... Create public key pair to connect with a client application

        Routing Options
        [r] ... Print routing table
        [a] ... Add new route manually
        [f] ... Flush routing table
        [w] ... Recalculate routing table by processing gateway information
        [x] ... Poll routing information from gateways and apply to local routing
            table now.

        [q] ... Quit
```

Options are categorised into:

- General options
- Member options

- Community options
- Service options
- Routing options

Each of them provides access to all important configuration facilities. Whenever required, files from the local file system may be inserted; however, since the maintain environment is connected to the G4DS backend, it is able to collect most of the required information from the local G4DS managers, or secondly, from connected nodes.

Several options within maintain are related to each other. For example, whenever a new community description is added to the system, gateway information for it should be processed and the routing table should be recalculated. These dependencies are resolved within the implementation; consequently, a required function will be called automatically by its host function after or before processing the function itself.

5.3.4. Access Control Implementation

The implementation of access control is based on policy files which are parsed at booting up time of G4DS and then applied to a permission matrix. The policy files have to be placed on the location as defined in the G4DS config file. By default the files are loaded, which are defined in the config file.

The access matrix is two dimensional. Each entry inside the matrix contains an ordered list of couples (action — reaction). Check section 5.2.4 for more information. Two separate processes occur for working with the access matrix, namely the loading of policies with its application to the matrix and the validating of a request originated from any location within the G4DS architecture. Either of them is explained in detail in the following sections.

Creating the matrix from policy files

The content of all requested policy files is loaded into memory using dictionary data structures. Afterwards the dictionaries are processed in order starting with the default ruleset as stated in the central config file with value POLICY_MAJOR_RULESET_ID. Hence, the order, in which the policy files are read and processed, does not matter (at least as long as no result set id is used several times, which would result in overwriting the value in the dictionary).

The processor iterates the list of rules in the major ruleset ordered by rule id (so all rules within the rulesets have to be kept in order regarding their rule identifiers). Whenever a rule has the reaction type 'direct', the value is put directly into the access matrix. (In fact,

for each possible combination of the rule for actor / target one item is appended to the list of rules for the corresponding couple.) Whenever the rule has the reaction type 'redirect', the processing is continued with the requested rule set immediately and, after finishing the redirected list, continued after the redirected reaction. Redirection are allowed in all rulesets; hence they may be nested.

Access the Matrix - validate requests

Whenever a request is performed against the authorisation engine, first of all, the authorisation controller is performing a look-up in the access matrix and this way loading the list for the requested couple of actor / target. This list is then iterated (by order as established at boot-up time) and the first rule hitting the requested action is taken. Regarding the action stored for this rule the function will return the appropriate value.

Roles for access control objects

As outlined in section 5.2.4, you may define roles for either of the three object types within access control. This way, rules do not need to be defined duplicately, since they may apply to a set of actors, a set of targets and / or a set of operations. Representatives of roles may be defined in two different ways: either by naming the ID of each representative or by providing a wildcard from the supported list of wildcards.

Supported wildcards are:

- '*' - all possible values
- 'authorities_communities' - all authorities for a community (only allowed for actor)
- 'authorities_services' - all authorities for a service (only allowed for actor)
- 'authorities_members' - the member for himself (only allowed for actor)
- 'Cxxxxxx' (in conjunction with type attribute 'membergroup') - All members of the community with ID Cxxxxxx
- 'Sxxxxxx' (in conjunction with type attribute 'membergroup') - All members of the service with ID Sxxxxxx

At boot-up time of G4DS access control the wildcards are resolved appropriately in respect of the targets as given by the rule. For example the combination of actor with role definition as given in Listing 5.6

Listing 5.6: Sample role definition

```
group
   rolename = communityauthorities
   representatives
      representative (* type='membergroup') = authorities_community
```

and the target as defined in the final rule definition as given in Listing 5.7

Listing 5.7: Sample rule definition

```
rule
   id = MY_ID
   comment
   actor (* type='role') = communityauthorities
   action (* type='action_id') = g4ds.control.community
   target (* type='community') = C002
   reaction (* type='direct') = allow
```

would create entries in the access matrix for all couples made up by the ID of any authority of Community C002 as actor and the ID of Community C002 as target.

Resolve nestings

In subsection 5.3.4 it was mentioned already, that by using redirection, rules may continue processing within another ruleset, and consequently in more rules. This way, there will be a cascading of rules, where each of them comes with its own definitions for actor, target and operation. Since G4DS access control resolves the rules into an access matrix, it must be able to handle different values for each of them from various rules.

To approach this problem we first of all have to visualise the logical relationship between the elements of the rule definition. Listing 5.8 shows the required AND operations for evaluating the rules; hence, to check whether a requested operation hits one particular rule:

Listing 5.8: AND crossing for elements of policy rules

```
actor(request) ELEMENT OF actorgroup(rule 1) AND
        actor(request) ELEMENT OF actorgroup(rule 2) AND
                ...
        actor(request) ELEMENT OF actorgroup(rule N) AND
target(request) ELEMENT OF targetgroup(rule 1) AND
        target(request) ELEMENT OF targetgroup(rule 2) AND
                ...
        target(request) ELEMENT OF targetgroup(rule N) AND
operation(request) ELEMENT OF operationgroup(rule 1) AND
        operation(request) ELEMENT OF operationgroup(rule 2) AND
                ...
        operation(request) ELEMENT OF operationgroup(rule N)
```

Inside each of the groups (actorgroup, targetgroup or operationgroup) there might be several representatives (check section 5.2.4 for details). Since, only one of the representatives needs to be hit at a time, there is a logical OR relationship between all the representatives of one group.

In practise, the OR and AND relationships are handled in two ways:

1. Resolving the values for actors and targets

2. Keep and store the values for operations in the matrix

This is an appropriate consens of resolving values beforehand (at G4DS access control boot-up time) and iterating them at request time (access control validating). We know about the finite collection of actors and of targets and are able to calculate all possible combinations between them. This way, for the first processed rule within the nesting the list of all available couples actor - target, which are addressed by the rule definition, are calculated and temporarely stored. Once, the processing is continued with a new rule, the same procedure is undertaken for its definition of actor and target. Due to the AND relationship between the two sets, we calculate the subset of couples, which are member of both sets and store this for now. This process is repeated for each rule in the nesting process; finally we end up with a set of actor - target couples, which are defined by all rules, we visited along the process.

In contrast, we are not able to determine all operation strings at boot-up time; the collection of operations is kept dynamic and may be extended by connected applications / services. Consequently, we have to picture the AND and OR relationships between operation strings in our access matrix model. Figure 5.3 in section 5.2.4 only visualises a simplified model of the topology; in practise the ordered list of action strings and corresponding reactions from Step 2 of the figure appears to be more complex than shown there. Each entry of the

action in each line of the list appears to be a list itself made up by action strings which are in AND relationship with each other, whereupon, these action strings are again not really action strings but a list of action strings, standing in an OR relationship to each other.

At validation time, the list for one combination of actor - target is still loaded from the access matrix. However, the iteration through the list becomes a bit more complex due to the AND and OR relations within each entry, which needs to be resolved here. For easy access of action strings dictionaries have been implemented to support this process.

5.3.5. G4DS and Communities

As stated in the introduction, members within G4DS are supposed to be grouped into communities, the so-called Trusting Communities (TC). Each member may be subscribed to several communities; this, however, does not include that each member may pass information from one community to another.

In order to carry out this action, they must possess the additional role *Community Gateway*, for which applies:

- Gateways are always defined for two communities and only one direction for passing messages between them.
- For each gateway constellation (source gateway, destination gateway and direction) there may exist several gateway instances with different members.
- The gateway member must be member of both communities, the gateway role is assigned for.
- The gateways are defined in the community descriptions (see section 5.2.3); thus, only community authorities may decide about their definitions. They are separated into incoming gateways and outgoing gateways; consequently, for each incoming gateway in community description A there must exist an outgoing gateway in community B and vice verse. (assuming, that A and B are the two communities involved)
- Each member may employ several gateways roles. Very likely will be the gateway role between two communities in both directions; however, a node may also pass messages between more than two communities.

The distribution of a message; hence, whether it shall be passed on to other communities depends on several descriptions and policies, which are:

- The community descriptions of the two communities involved

- The access policy, which is usually coming with a trusting community description
- The access policy of the local node
- Information from the connected application, how to handle the given piece of information (which means, distribution information on top of the receivers address)

Once the passing of messages involves a higher number of communities, it shows more and more similarities with the routing process employed in the Internet. Each node within G4DS is maintaining a routing table, which is updated frequently using a polling mechanism. Each entry within the routing table is made up by the following values:

- ID of the Source Community
- ID of the Destination Community
- ID of the member for the next hop on the route to this community
- ID of the community to be used to connect to the member on the route towards the final community
- Information about costs, which are measured in number of hops on the way to the final destination

The coming two sections break down the problem into the two areas, namely the routing (passing) of messages in regards to a routing table, and secondly the installation and maintenance of the aforementioned routing table.

Routing of messages

The routing of a message is quite similar to the routing mechanism within the Internet; hence, as employed for the TCP/IP protocol stack (Stevens (1994)). It starts with the sender of a message, which will perform the following actions:

- The sender checks, whether it is in the destination community of the message itself. (The destination community is defined by the connected application, if not given, one will be calculated by the routing system)
- If the sender is member of this community, it will deliver the message directly to the final receiver

- In case of a mismatch regarding the communities, the routing engine checks whether there is a route available towards the final community (including permission checks against the local access control)
- If no route may be determined, the sending of the message is cancelled and an error will be reported to the connected application
- If, however, a route may be calculated, the actual message is wrapped into a G4DS routing message and passed through the G4DS control system, which will send it off to the next hop on the way towards the final destination.

Each receiver of a message may determine whether the incoming message is a routing message by examining the type of message within the G4DS control system. If the incoming message is a directly sent message, it is passed on directly to the connected application (in case of a service message) or the corresponding control sub-system (in case of a G4DS control message). If this is not the case, however, the following steps are performed:

- Check the community of the incoming message and attempt to find a value within the routing table towards the final destination community (if there are several entries in the table, pick the one with the lowest cost)
- Check the acquired route against the access control
- If the acquired route does not involve any further hops, the message is wrapped into a routing message with destination of the final receiver of the message
- If, however, another hop is required, the message is wrapped into a routing message with destination of the next hop and destination community as stated in the routing table entry

Using this infrastructure, applications may specify destinations for their messages all over the G4DS topology; the local node is the starting point of routing by sending it off to the right community gateway. From there it will be routed through the entire network until it reaches its final destination.

Maintain the routing table

In its basics the routing mechanism of G4DS is grounded on the Routing Information Protocol (RIP) (Hedrick (1988)). The so-called gateways are establishing connections between each other frequently in order to exchange their routing tables. When bringing up a G4DS node for the very first time, it will perform the following tasks:

- Iterate all communities, the local member is member of, and create a value in the routing table with cost 1 (source and destination community are equal)
- Iterate all known gateways (which were assembled by processing community descriptions in an earlier stage) and create a value in the routing table with cost 1 (source and destination community as stated in the gateway)
- Again iterate the list of all known gateways and pick the ones, for which the source community is the destination community of one of the aforementioned two steps and create an entry in the routing table with them using value 2 for cost

After this initial setup, the dynamic routing will kick in on this node and, this way, it will connect to all directly connected gateways known on this node. In detail, this process is made up by the following actions:

- Determine a list of gateways by evaluating gateway information, which was created in an earlier stage by processing community descriptions
- Connect to each of these gateways and request a copy of their routing table (this request is encapsulated in a G4DS control message subsystem routing controller)
- Decode the XML encoded routing tables of the replies (by using functionalities of the XML Wrapper) into an easily accessable dictionary format
- Iterate the dictionaries and compare there content against local routing information and, this way, add or update entries in the routing table

The behaviour of the routing system may be influenced by settings provided in the configuration files. For example, the time interval between two poll procedures may be defined or the entire dynamic routing may even be disabled. The status of the current routing table may be shown using the maintain environment coming with G4DS (see section 5.3.3 for details).

5.3.6. Service Integration

The final aim of G4DS is the use with applications, the so-called services. In section 5.2.3 it was explained already that for each service there must exist an XML formatted service description, which must be added to the local G4DS system through the maintain interface (section 5.3.3). The following two sections will explain, how messages for services are handled within G4DS, and furthermore, which way a service may connect to G4DS and make use of it in order to communicate with other members of this service.

Service identification and dispatching of messages

It was stated already that G4DS separates two types of messages; namely (G4DS internal) control messages and the service messages. By processing the message type of an incoming G4DS message, the global message dispatcher is able to determine the type of a message and, in case of a service message, passes it on to the dispatcher of the ServiceIntegrator.

The service integrator thereupon will perform server al actions:

1. Extract the service id from the message and test against the access control, whether this message is allowed on this node
2. Perform application layer access control. By providing an action string to G4DS when passing on a message, services may make use of G4DS access control facilities. This way, messages are filtered on G4DS layer before they are even passed on to the application. This behaviour is explained in more detail for the IOIDS service in section 6.3.3.
3. Check, whether any client application has connected beforehand for the service with the given service id
4. Looks for the callback function for the connected client application and passes the message on to it (using the FIFO mechanism as described in the next section).

For outgoing messages an interface is provided as well. It assembles the corresponding G4DS service message and passes this one on to the global outgoing message handler, which thereupon will take care of the final delivery.

Connect applications to G4DS

In order to allow several applications to use G4DS at the same time, a client / server approach has been put into place. This is implemented using FIFOs. Consequently, it is compulsory, that one starts the server before they can use G4DS with any application. In more detail the following actions have to be performed in order to make use of G4DS functionality from within any application.

Start G4DS Backend The easiest way to start the server is the use of the provided init script (g4dsrc). During the installation process, it is copied to the folder "/etc/init.d". Start the server with the command:

```
/etc/init.d/g4dsrc start
```

You may stop it later on using

/etc/init.d/g4dsrc stop

Any time you may check whether it's running using the command:

/etc/init.d/g4dsrc status

Whenever it is required to start the server manually, the python script "g4dslistener.py" has been installed to "/usr/sbin" during the installation process.

Connect with client application G4DS implements asymmetric key authentication for clients to connect against the G4DS Backend. Hence, before any application connects to the grid system, a private / public key pair has to be created. In order to do so, the maintenance environment (see section 5.3.3) of G4DS has to be started. The following instructions have to be followed:

python maintain.py # this starts into interactive mode

Choose option '1' here and create a new public / private key pair. Export the private key to a location within your local file system, where one can access it from the application later on; add the public key to the list of authorised services.

Ones, the server has been started and the key has been created, the application may connect to the grid system. Basically, the commands as provided in Listing 5.9 are to be used within the Python code:

Listing 5.9: Connect with Application against G4DS

```
path_private_key = 'your_location_of_key'
service_id = 'S1234567890'                                    # ID of your service

from g4ds.g4dsservice import G4dsService          # g4dsservice provides the
    interface for the client applications
gs = G4dsService()
gs.connect(service_id, None, path_private_key, incomingMessageProcessor)

... do lots of stuff here

gs.sendMessage(destinationMemberId, destinationCommunityId, message)        #
    sending one message through G4DS

gs.disconnect()                                                  # disconnect from
     the g4ds backend

def incomingMessageProcessor(message):
    pass        # do whatever you want to do with incoming messages from G4DS
```

5.3.7. Message identification and job implementation

In certain cases there will be a need to process knowledge from several locations from within the G4DS topology. This process might be time consuming and involves the requirement of linking information to each other.

Therefore messages must have a unique ID in order to allow referencing of them later on first of all. Furthermore, the time consuming overall process must not block the entire node from serving other requests. The following two sections are dealing with these two concerns.

Message identification and message replies

In order to identify messages appropriately a unique message id is generated before a new message is send off. This responsibility is overtaken by the function generateId inside the tools module. At the current stage only a random number is generated to serve this job. This message id is always sent as part of a G4DS message.

Whenever a message shall be sent as a reply to an earlier message is must reference its message id. The GlobalOutgoingMessageHandler supports this functionality by providing a parameter in the interfaces of the sending message functions. (It will then use wrapper functionality to include this information in the XML message finally.) The handlers for both, service and control messages are providing an interface as well to put references into messages, and will pass the id on to the GlobalOutgoingMessageHandler.

Job context and job continuation

As stated before, everyday work of G4DS includes actions incorporating several time consuming actions themselves. In order to prevent the system from being unreachable we have to introduce different threads at this point.

There is not a real job scheduling in place for G4DS, instead for certain functions, which are thought to be time consuming at execution time, facilities have been provided to run them in a dedicated thread. This way, these functions look as described in listing 5.10.

Listing 5.10: Background Processing and Job Locking

```
def bigFunction(par1, par2, inBackground = 0):
  if inBackground:
    import thread
    thread.start_new_thread(bigFunction,(par1, par2, 0))
    return

  msgId = getMemberController().requestMemberDescription()
  jl = JobLocker()
  getJobDispatcher().addJob(msgId, jl)

  # here we wait now until any message arrives referencing us

  message, args = getJobDispatcher().getMessage(msgId)
  if args['key'] == any_value:
  # further processing
```

By default the functions are not put into background. Listing 5.10 visualises the way to do so using the variable *inBackground*. If this variable is set the function will start itself again in a new thread and return directly afterwards in the calling thread. The parameters are exactly passed to the new function inside the thread as they were received in the first place (all but the inBackground variable of course).

Listing 5.10 also provides information about the job locking process. Job locking needs to be utilised whenever some information has to be downloaded before a job can be continued. Any paused job is always identified by a message id. This way, it is easy to identify the corresponding job for the incoming message dispatcher just by evaluating the value of the reference id (as discussed above in Message identification and message replies). The controller for this particular message performs a look-up in the dispatcher table for the extracted ref id and will try to resume the connected job.

Listing 5.11: Example of continuation for a job

```
def CommunityController.dispatch(msgId):
  from messagehandler import getMessageContextController
  msgref = getMessageContextController().getValue(msgId, 'refid')
  if msgref:
    print "\n\tAttempt to resume Job %s" %(msgref)
    from runtimecontroller import getJobDispatcher
    args['success'] = success
    getJobDispatcher().resumeJob(messagereference, data, args)
    else:
        # other community controller processing
```

An example is shown in Listing 5.11, which demonstrates the dispatching procedure within

the G4DS configuration controller - subsystem community controller. Using the message context facilities it is determined whether this message is in reply to any message sent from this node. If this is the case, the extracted data will be passed to the job dispatcher, which will thereupon continue the corresponding job. The last three lines of Listing 5.10 show, how exactly this data is collected from the job and processed further in there. Once the community controller has resumed a job, it will not carry on with any processing of the incoming message.

The community controller just serves for this example. Exactly the same way other dispatchers are carrying out their work in order to support job continuation.

5.3.8. Logging Facilities

A G4DS instance is running in several threads, which makes it very hard to pass errors, exceptions or runtime messages to any connected user or application. Furthermore, most of the times the G4DS backend will run as a Linux service (see Section 5.3.6); consequently, there is no direct user interaction at all. However, the user / administrator should always be in the position to track movements within G4DS concerning the local node. This way, a logging mechanism has been put into place.

Logging implementation overview

G4DS logging enables the G4DS system to log into two different targets:

- A G4DS unique log file (with G4DS syntax)
- Linux / Unix common syslog facilities (Lonvick (2001))

The G4DS unique logging files look similar to the well known Linux syslog files; hence, each entry in there is equipped with a date-time stamp and a logging message. Additionally, an action id is provided, which enables easy processing and filtering of log information without evaluating the log message text itself.

In order to put G4DS logging output into the syslog facilities, a syslog service has to be started on the local machine. G4DS logging connects against this instance and will pass on all messages equipped with the service identifier *g4ds*.

Settings for G4DS logging may be done in the global configuration file for G4DS. These allow to specify file names for G4DS file logging, to change the service identifier for syslog logging and to adjust the log level to the local needs.

Logging level and further settings The entire logging implementation is encapsulated in the module *g4dslogging*. The 6 different log levels are defined there in detail. In fact, each log-level is defined by a dictionary, which includes the action ids to be logged in this level. Practically, the log levels represent:

- Level 0 - Critical errors only, logging system status messages
- Level 1 - Communication errors
- Level 2 - Incoming / outgoing messages
- Level 3 - Message details (msg id, sender, control sub system, service, ...)
- Level 4 - *currently unused*
- Level 5 - All messages

Contribute and access logging information

A new message may be inserted into the log system by using the logging system singleton access and the method newMessage, which looks like code sample in Listing 5.12 (taken from the incoming message handler):

Listing 5.12: Example for generating log information

```
from g4dslogging import getDefaultLogger, COMMUNICATION_INCOMING_MSG,
    COMMUNICATION_INCOMING_MSG_DETAILS
getDefaultLogger().newMessage(COMMUNICATION_INCOMING_MSG, 'New incoming message')
# ... process more information here
getDefaultLogger().newMessage(COMMUNICATION_INCOMING_MSG_DETAILS, '-- MSG ID %s | '
    SENDER %s' %(mid, senderid))
```

The log information may either be accessed from the standard syslog log files (if syslog is enabled) or by accessing the G4DS unique log file (location as defined in global G4DS config file). If G4DS file logging is enabled, one may also use the *maintain* environment (see section 5.3.3) for accessing the latest G4DS log entries.

5.3.9. Modularity and Extensibility

Controllers, Interfaces and Implementations - integrate algorithms and protocols Up to now there was only mapped the knowledge about where protocols have to be used which way, which keys have to be employed and what algorithms have to be used for encryption,

decryption and signing. The implementations of the algorithms and the protocols however; and how to handle and use them has not yet been addressed.

For these purposes a so-called controller concept was introduced. Either of the two, protocols and algorithms, has got its own controller; however, both them are working exactly the same way. A major benefit of this approach is the flexibility. New protocols or algorithms may simply be added to the G4DS infrastructure by writing a new module (implementing a pre-defined interface) and registering the implementation in a config file. The following section explains the concept for protocols in detail; changes applied for algorithms are discussed afterwards.

Controller for Protocols

G4DS is supposed to support a variety of communication protocols. The approach for integrating them must be adaptive; hence, new protocols should be integratable very easily. The initial version of G4DS comes with support for SOAP connections and communication through plain TCP / IP sockets.

The central controller for the protocols is placed in the G4DS main package. It is represented by a class ProtocolController located in its own module named protocolcontroller.

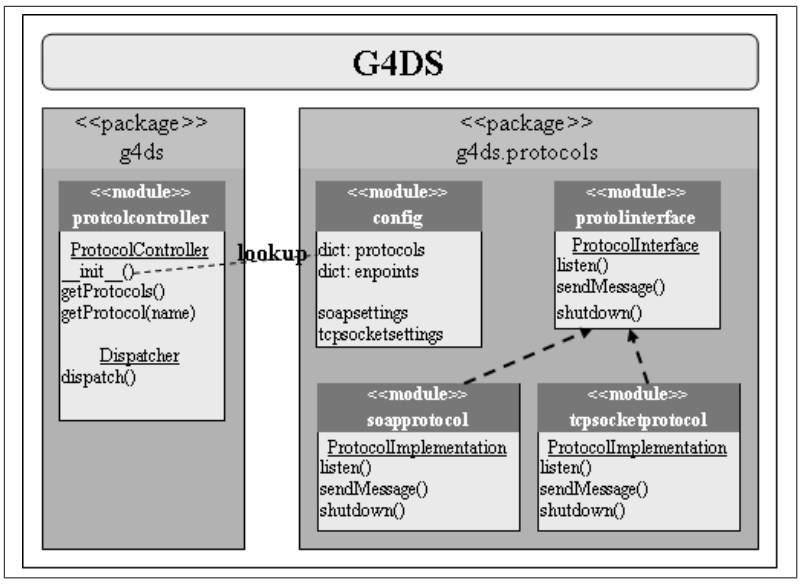


Figure 5.5.: Controller for Protocols

The extensible bits, however, are located in their own package called g4ds.protocols. First of all, in there a module protocolinterface is provided containing a class named ProtocolInterface. This class has no implementation; instead it just provides an interface with names of functions

and their signatures, which have to be overwritten by any implementation of a protocol.

One of these protocols is the soap protocol. A new module should be created for each protocol implementation mirroring the name of the protocol; however, there is not really a naming scheme for this. Afterwards a class named ProtocolImplementation has to be created in there which must inherit from protocolinterface.ProtocolInterface and overwrite the functions defined in there. The listen function hereby must create its own thread; otherwise it would block the entire application. After starting the new thread it should listen for incoming network connections. Settings therefore may be given inside the config file, where a section for each protocol implementation may be created.

Finally, each protocol implementation has to be registered with G4DS. This is performed using the config file inside the g4ds.protocols package. A dictionary is in place, whose key is the name of the protocol and the value the module. This information is important since they are used at initialisation time of the ProtocolController for bringing up the protocols and making them listening for incoming connections.

Another dictionary named endpoints is located within the config file. It is only used before first bringing up of G4DS and therefore for the initialisation of the local database. In fact, it tells the initialiser how the addresses for the certain protocols are looking like. The keys for this dictionary are again the names of the protocols; the values are now a string representing the protocol specific address. For example, a SOAP address should be a URL, whereby the TCP socket address should be a combination of DNS name and port number.

The ProtocolController at initialisation time will gain the list of protocols from the config file in the g4ds.protocols package. By iterating the protocols dictionary it may gain the names of the modules used for the implementations and due to the common class name and method names and signatures may initialise the protocol implementations. For later use of protocol implementations functions are provided for both getting a list of all names of usable protocols or directly access a protocol implementation.

Controller for Algorithms

The mechanism for accessing the implementations of algorithms is following exactly the same approach. A controller named AlgorithmController is provided in the main package g4ds in its own module algorithmcontroller. It comes with an interface for accessing the implemented algorithms and for producing a list of names of available ones.

A common interface has been defined for algorithms supporting all the features of an Public Key Infrastructure (PKI) algorithm; namely encryption, decryption, message signing and message validation. At the initial state two implementations for algorithms are coming

with G4DS, one for the RSA algorithm and another one ElGamal.

An additional feature has been introduced for the AlgorithmController, namely the one of loading private keys into the algorithm implementations. This is reasonable for avoiding the passing of private keys throughout the entire code all the time since a reference to the private key is maintained inside the algorithm implementations.

5.4. Conclusion

With Grid for Digital Security a robust, secure and reliable communication platform has been presented, which may be used by a variety of knowledge sharing applications. The persistence in the database backend allows every node within the grid to maintain its own view of the network. The implemented concept of communities supports very well with the attitude of establishing trust relationships between parties. A high grade of adaptability has been achieved by implementing XML based configuration facilities. The major issue of security has been adopted in an extensive manner – the utilisation of a private / public key infrastructure supports with encryption and authentication and is even able to adopt new algorithms easily. An access control list mechanism prevents any data to be accessed by non-authorised parties and is even useable for connected applications.

The major point of G4DS is keeping the responsibility to the user, meaning that the local node itself decides about which party they trust for which kind of operation on which kind of data. By providing a module for knowledge services to connect against G4DS this operation may be performed very easily. Last but not least allows the generic and adoptable implementation of G4DS the utilisation for a wide range of services, only assuming they deal with the sharing of knowledge within groups.

Inter-Organisational Intrusion Detection (IOIDS) is the application running on top and making use of Grid for Digital Security in an extensive manner. Its architecture and mode of operation is explained in very detail in the upcoming chapter 6. Afterwards, the execution of experiments and a critical analysis of the results will prove the applicability of the combination of G4DS and IOIDS for the given problems and objectives.

Chapter 6.

Inter-Organisational Intrusion Detection System (IOIDS)

6.1. Introduction

In chapter 3 the idea and requirements for an architecture for exchanging security related incident information have been discussed in detail. Chapter 5 provided very detailed information about the implementation of the subjacent data exchange topology. The missing bit in the overall architecture is the application running on top of this network infrastructure, which will be discussed in detail in this chapter.

The application for incident exchange has been called "Inter-Organisational Intrusion Detection System" due to its nature of exchanging security related incident information across organisational boundaries. It implements all issues directly related to the distributed intrusion detection feature and, extensively, makes use of the communication facilities provided by the subjacent knowledge grid architecture "Grid for Digital Security".

Firstly, an overview is provided of the overall design of the IOIDS application; about its modules and their interaction. Afterwards, more attention is drawn to the design features and issues such as complete data flow between two IOIDS instances and database backend implementation are discussed in great detail. Finally, major components of the implementation are introduced, which provides an insight to facilities such as interaction with the Grid system, employment of access control and its integration with G4DS access control (Section 5.2.4) and, finally, handling and converting between internal datastructures, which is stressing to some extend the implication of XML data formats (Section 2.6.1).

6.2. Design

The Inter-Organisation Intrusion Detection system architecture has been implemented in a modular way and is communicating with a variety of components for carrying out its work.

Figure 6.1 visualises a high-level description of theses components and the way they are linked to each other.

6.2.1. Overview of design

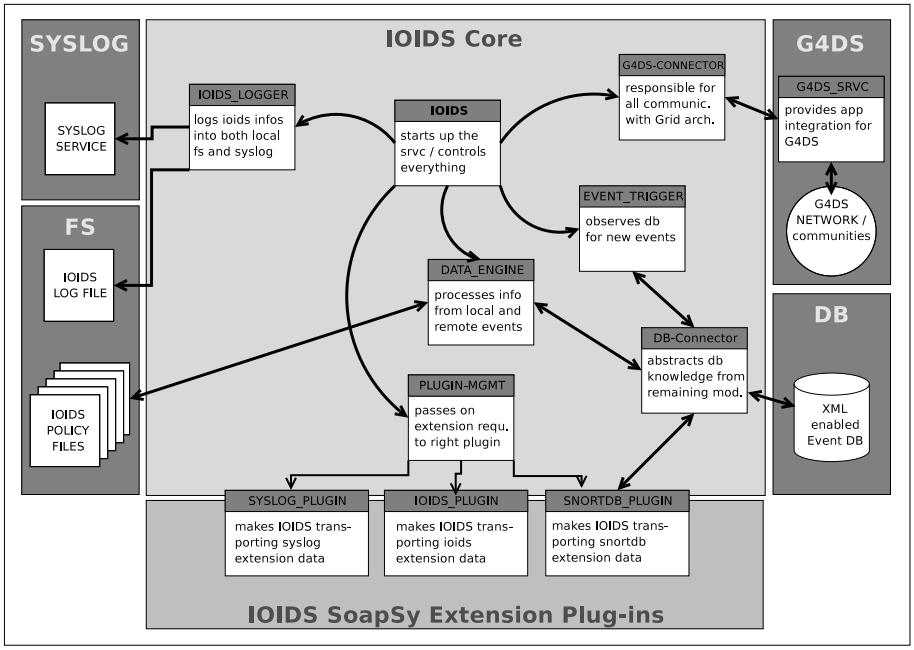


Figure 6.1.: Overview of IOIDS architecture

Starting with the core-application of IOIDS, it is the part to keep the components together and integrate with connected components. The following list provides an overview of these components and a brief introduction to their responsibilities:

G4DS – The Grid for Digital Security (G4DS) communication platform. Using a G4dsConnector within IOIDS the system can make use of the facilities provided by G4DS. All traffic towards and from G4DS must be passed through this module. More details for interaction with G4DS is provided further down in this chapter in section 6.3.1.

DB – The backend persistence for IOIDS is implemented using a unique XML based database management system. All access to this database has to be performed using the module DBConnector. The database layout and all concerns for retrieving and inserting data from and into this database are discussed in great detail in section 6.2.3 within this chapter.

Syslog – IOIDS is supposed to run as a service rather than as an interactive application. However, users of the system must be able to trace the system's behaviour and check for certain events. Consequently, a logging mechanism has been put into place, which, most conveniently, is able to integrate with the system event logging system *Syslog*. A very similar approach has been introduced for G4DS (section 5.3.8), some more details are discussed in section 6.3.5 further down in this chapter.

FileSystem (FS) – Some information for IOIDS is stored in the local filesystem. Besides the extension for the aforementioned logging facilities into a dedicated local log-file the access to the file system is almost exclusively restricted to read access and supports the configuration of IOIDS with configuration files and policies. The configuration itself is a plain text-based configuration file and does not require further discussion, the utilisation and distribution of the policies, however, is discussed in section 6.3.2 in the implementation section of this chapter.

IOIDS SoapSy Extension Plug-Ins – The SoapSy database (as discussed in section 6.2.3) provides facilities for a variety of event generating applications or tools. In order to support the distribution of this knowledge throughout the communities using IOIDS, it must understand the structure of each of the extensions. The problem in more detail and the solution to tackle it are discussed as part of the SoapSy DB introduction in section 6.2.3 within this chapter.

Whenever IOIDS is started up, it initialises all the connected services and components using its modules (indicated using curved arrows within figure 6.1). However, information between these modules may be exchanged directly (rather than the diversion through the IOIDS module). Straight directed arrows in figure 6.1 indicate these relationships.

6.2.2. Flow of data

A variety of components are working as a unit in order to make IOIDS working. In order to enhance the understanding of the complex overall system, the following ordered list of actions examines the process of sending an event from one node to another.

The following assumptions are made before the event can be triggered and sent:

- The Grid for Digital Security (G4DS) communication platform is running and IOIDS has connected successfully.
- The local node is member of at least one G4DS community (which supports the IOIDS service) and knows about several nodes within that community, which are also running

the IOIDS service and is able to exchange messages with them (hence; reach their endpoints).

- The backend XML database management system is running and IOIDS has successfully connected against it.

Assuming our system is in a state, which satisfies all the aforementioned requirements, it is now able to exchange knowledge with other nodes. Basically, there are two different kinds of messages exchanged within IOIDS, namely:

- IOIDS knowledge requests (a message, requesting more information from other IOIDS nodes, hitting certain parameters)
- IOIDS information updates (a message, informing other nodes within the IOIDS network to update their information repositories)

Both them are discussed in greater detail in section 6.3.2. For now, we focus on an information update only, the knowledge request, however, involves the same components and does not differ in the way a message is passed through the IOIDS system.

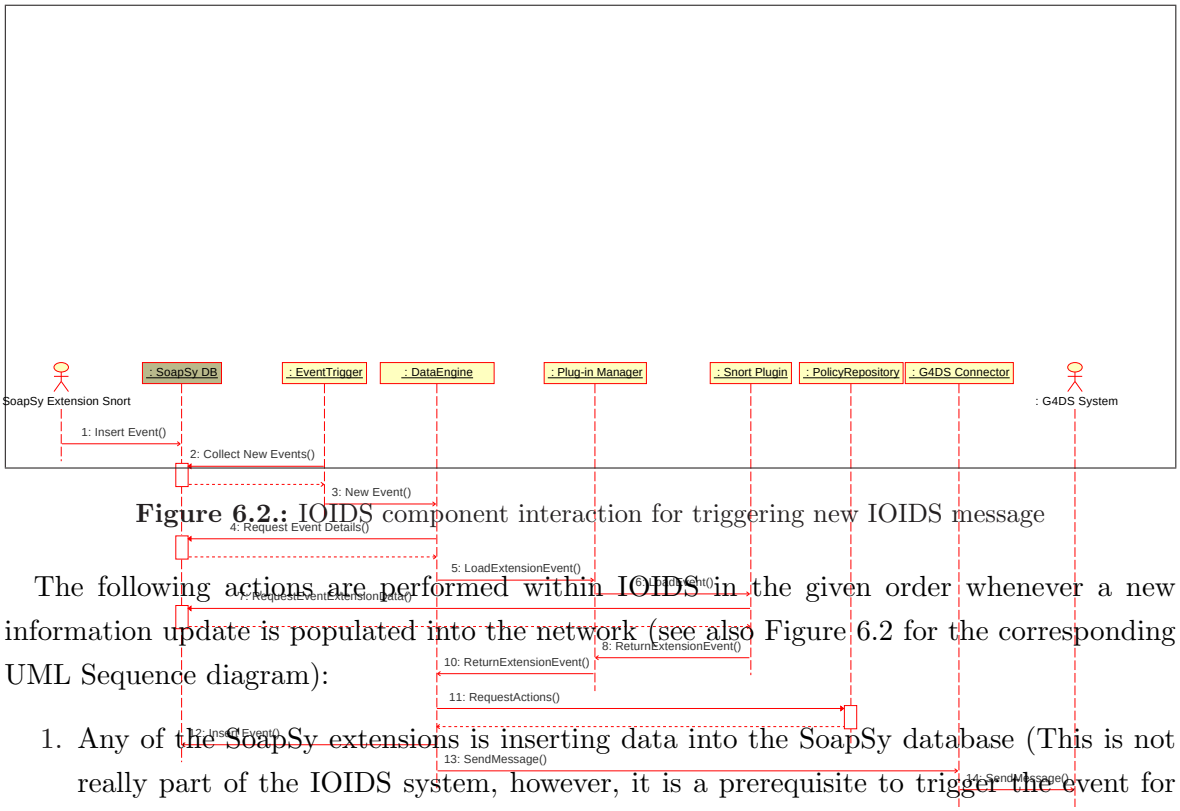


Figure 6.2.: IOIDS component interaction for triggering new IOIDS message

The following actions are performed within IOIDS in the given order whenever a new information update is populated into the network (see also Figure 6.2 for the corresponding UML Sequence diagram):

1. Any of the SoapSy extensions is inserting data into the SoapSy database (This is not really part of the IOIDS system, however, it is a prerequisite to trigger the event for IOIDS).

2. The IOIDS *Event Trigger* has been observing the database constantly using the *DB Connector* and becomes aware of new events and informs the data engine about this occasion.

3. The IOIDS *Data Engine* gathers all available core information for the given event and passes a request to the plug-in-management to gather extension information for the event.

4. The IOIDS *Plug-in Manager* attempts to load the corresponding plug-in for the given extension and, if successful, passes on the request to it.

5. The IOIDS Plug-in loads the extension information from the XML database and passes the results to the data engine through the plug-in manager.

6. Now it is up to the data engine to assess the data and initialise appropriate reactions. These reactions are defined in the policy files, which are loaded and parsed. If requested, an IOIDS event is created (equipped with a classification and destination community) and passed to the IOIDS DB Connector for insertion into the local data repository.

7. Again, aligned to the defined policies, the data engine may decide to pass the message on to certain nodes within the grid system (G4DS). For these reasons, the new event (optionally including related events such as the initial event triggering the whole process) are encapsulated in an IOIDS message and passed on to the *G4DS connector*.

8. The *G4DS Connector* thereupon equips the message with additional status data and is passing it on to the connected grid system.

From here, control of the message is taken over by G4DS. This process has been discussed in great detail in the G4DS chapter in section 5.3.2.

After passing through the G4DS system (and distribution to its defined receivers), the message will be dispatched on the receivers side within the G4DS system and passed on to the connected IOIDS service. On the receivers side, the following actions are performed with the incoming IOIDS message (see also Figure 6.3 for the corresponding UML Sequence diagram):

1. The grid system dispatches the message as a service message and pushes it up to the IOIDS system regarding its service identifier.

2. The dispatcher as part of the IOIDS *G4DS connector* receives the message and after extracting the status information it informs the *Data Engine* about the new incoming event and about its nature of an information update.

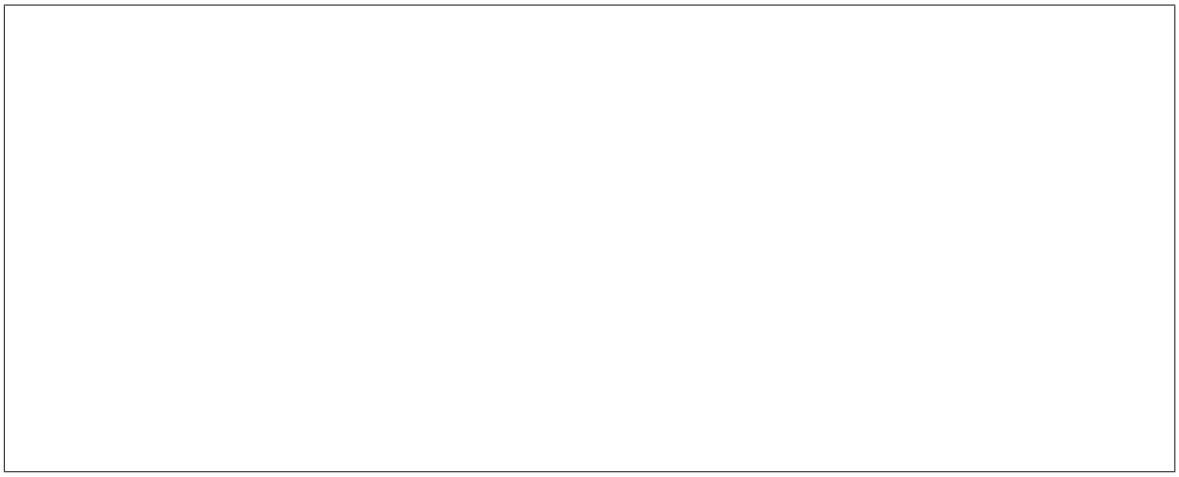
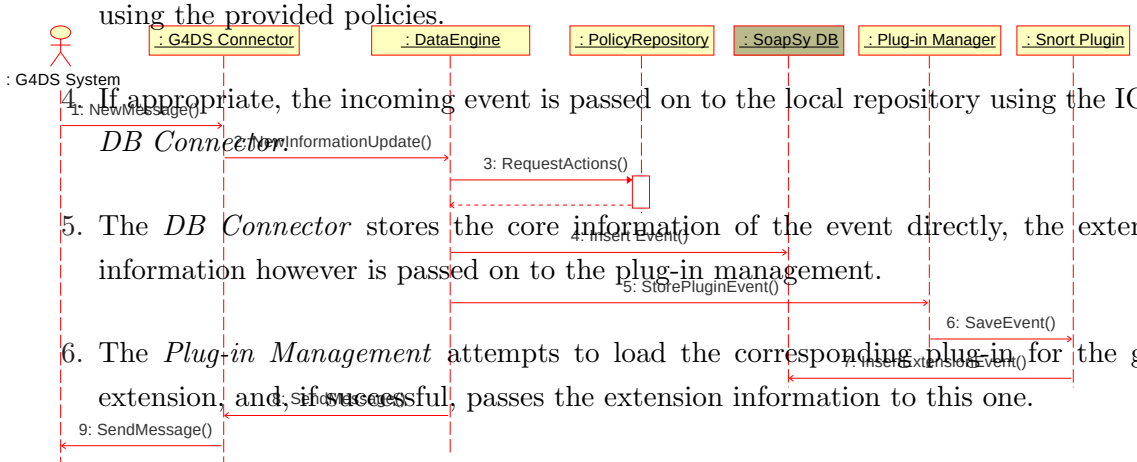


Figure 6.3.: IOIDS component interaction for receiving new IOIDS message

3. The *Data Engine* thereupon parses the message and determines appropriate action(s) using the provided policies.



4. If appropriate, the incoming event is passed on to the local repository using the IOIDS *DB Connector*.

5. The *DB Connector* stores the core information of the event directly, the extension information however is passed on to the plug-in management.

6. The *Plug-in Management* attempts to load the corresponding plug-in for the given extension, and, if successful, passes the extension information to this one.

7. The corresponding plug-in connects to the database through the IOIDS *DB connector* and using its intelligence about the structure of the extension data inserts it into the database.

The *Data Engine* on the receiver side might trigger further actions besides the local storing. This way, it might take the new knowledge, correlate it with other event information and pass on a new event to a community / several communities. This way, the whole process may run again and again. Consequently, attention has to be paid to the avoidance of circulating information within the IOIDS system. The IOIDS implementation takes care of this matter by introducing unique IOIDS message identifiers, which are explained in detail in section 6.3.2 within this chapter.

6.2.3. Database backend

In the review of existing technologies in section 2.8.1 several approaches for making information persistent have been introduced and special focus has been put on the variety of existent database technologies, which are:

- Plain relational database
- Object Oriented database
- XML based database

First of all, each of them is able to support the persistence for the IOIDS architecture in general. However, the following issues were to be kept in mind for choosing one out of the available technologies:

- Data encoded as text of various lengths, values in date / timestamp formation and numeric information should be stored efficiently
- The length of text fields should be supported up to a high value (approx. 100,000 characters) [1]
- Data should be accessable easily, and if possible in large chunks; hence, complex data sets as a unit rather than each to be fetched separately and to be assembled on db client side

When comparing the attributes of the database types (2.8.1) against the given issues, the choice for one of the technologies can be made:

- The plain, relational database supports very well with storing and fetching information encoded in text (of various lengths), timestamps as well as numeric information. Also the issue of efficient storage of long character sequences is supported very well (for example using the wellknown datatype VARCHAR). Finally, it is able to support the handling of structured information; however, there are some shortcomings for their utilisation:
  - The assembling of the database query (in Structured Query Language (SQL) (Groff and Weinberg (1999))) may either be performed in form of nested select statements of in form of joins, which appear to be rather complex and difficult in syntax as the complexity increases.

[1] This is required due to the need of storing the entire payload data of events

  - In either of the two aforementioned ways of querying the database, the result will only be "one-dimensional"; hence, all the information from different relations will be provided as one result and separation of these are to be performed on db client side.

- The object oriented database would perform exceptionally well with the storing and accessing of structured data and is of course as well able to support all the required simple data types such as text-encoded data, numeric data or timestamps. Also, the handling of text encoded data of large length does not appear to be a problem for this kind of databases. However, the employment of an oo database would seem to be rather artificial since we are dealing with structured data such as events (check section 6.3.4 for details) rather than so-called objects in its original intention.

- An XML-based approach for accessing the database depends first of all on the subjacent database it is wrapping on concerning its supported datatypes. Assuming, an appropriate candidate has been chosen at that point, the support for all required datatypes with the certain needs in size in particular is not of any concern anymore. A major benefit of XML databases is their ability to handle complex data; meaning structured data with relations between entities. However, the datatype, they are working on, is of course XML, and again, the db client is in charge to formulate its query in XML firstly, and parse and extend its data from the XML encoded response secondly.

Comparing the shortcomings of the latter two approaches with their advantages in handling complex data, it was decided that it is worth the effort to implement one of the special database access encodings (either object oriented or XML based) - finally, the plain relational db query strings have to be assembled too and their response parsed and information extracted as some point.

The decision was finally made for the XML based approach due to the following reasons: Firstly, XML has been widely employed for this project already and several parsers and wrappers have been produced in the meantime, which supports the assembling of queries and the parsing of replies. Secondly, there has been developed a light-weight XML based RPC access methodology for databases within the research group (Xynos (2005)) which provides very simple access facilities to any subjacent database (see section 6.2.3 for details), which brings up the third justification for the choice: due to the implementation of the aforementioned database within the research group, a number of extensions for a variety of event generators have been developed and are inserting event data into the database. This supports the objective of handling information from a variety of third party event generators (section 3.3.4) and is discussed in greater detail in section 6.3.2.

Database design

After the decision has been made for a database access approach, the design for the database itself has to be developed. The following issues have been arising when dealing with this concern:

- The database design must support the persistence of computer security incident event information, including information about their involved parties, the type of the attack, date and time information and much more. A great collection of information involved in these environments may be found in (Debar et al. (2005) and Demchenko (2003)).
- The information should be structured and broken down into its components. (e.g. different relations for the event itself and its information about the victim, observer, etc.)
- Relations should be able to be maintained between several events. (e.g. one event might be derived from several other events, which should be able to be pictured in the database design.)
- IOIDS is supposed to support a wide range of third party event generators. All information, which may not be able to be normalised into the db event core system should be able to be stored in some extensible area within the database. This access should be provided easily and efficiently and must allow the maintaining of links between the extension information and the actual event.

In the information security research group within the univerity many research has been undertaken into the areas of unification of security related incident event data over the past years. (Avourdiadis and Blyth (2005a,b)) Many efforts have been put in the development of a database design, which is able to store information from a variety of data sources on the one hand, but is still kept simple on the other. The following section explains the idea of the new approach called SoapSy to some extend and mirrors its capabilities against our issues.

Overview for SoapSy database

According to (Avourdiadis and Blyth (2005b) *SoapSy*) is defined the following way:

> [SoapSy represents a] process of data unification and fusion into an abstract relational schema. The function of this abstract schema is to convert event related data into information relating to security incidents. A security incident is simply

> a representation of a grouping of event that is characterised by a common set of attributes.

This way, SoapSy provides a central data repository for logging security incident events from several locations from a variety of different event logging applications into it. The third party event generator has to normalise its event information in a way, that it may be integrated with the SoapSy event data model. A complete copy of the SoapSy database design is provided on the resources CD.

One can see, that the database design is made up by three dedicated parts, namely:

- The SoapSy Core
- The SoapSy Extensions
- The SoapSy Abstract

The *Core* part of the database design is holding the information all events have in common, such as event source, destination, target and the like. All applications inserting data into SoapSy must normalise their event data in a way that they provide the SoapSy core information. The *extension* part of SoapSy in contrast is supposed to support the storing of data of the applications, which can not be covered in the core part. Consequently, for each type of event generator (such as Syslog, Snort or Windows Event Log) there will exist an extension in the SoapSy extension part. The (extension) type of an event may be determined by the field *eventtype*. Finally, the *abstract* part of the SoapSy database layout introduces the concept of incidents. Following (Avourdiadis and Blyth (2005b)) their responsibilities are:

> Incidents are introduced as a mean for describing events that are related to each other in such a way that can describe a series of actions towards the successful or not completion of a goal.

At the current stage, the abstract part of the SoapSy approach is not deeply developed and there is not known any application, making use of this part of the database. Later developments, however, may employ decent data merging and data mining technologies in order to incorporate with relations provided in there. For now, they are not of any use for the IOIDS system since we are dealing with single events only; consequently, the abstract part of the SoapSy has not been employed for IOIDS.

Considering the issues discussed at the beginning of this section, the SoapSy database design suits very well the needs for persistence of IOIDS event data. Last but not least the ability to support a variety of third-party event generators easily encourages the employment of this approach.

SoapSy extensions

SoapSy extensions have been mentioned in the previous section when talking about the design of the SoapSy database. However, since IOIDS needs to store additional information for each event in the SoapSy database as well, some more detailed information will be provided in this section.

As stated before, SoapSy extensions are supposed to store all the information for one event type (for one event generating application), which can not be covered with the core of the SoapSy DB layout. In the experiment for IOIDS this idea is explained in greater detail for a SnortDB event logging system, whose event information is taken and transformed in order to insert into the central SoapSy data repository. (see section 7.3.3)

SoapSy extension for IOIDS event information IOIDS itself deals with information for each event, which can not be integrated with the core of SoapSy. These include:

- The community, this event belongs to (check section 6.2.5 for more information)
- The classification of the event (check section 6.2.5 for more information)
- The timestamp of receiving the event (rather than occurrence on the observer's and destination's side)
- Initial source of the IOIDS event
- Direct sender of the event
- Relations with other events within the SoapSy system

For these reasons, IOIDS maintains its own extension within the SoapSy DB Extension sub schema. The following relations are introduced within this extension to support this task (see also Figure 6.4):

IOIDS-CLASSIFICATION maintains a normalised list of IOIDS classifications with their codes and descriptions

IOIDS-PEER is a result of db normalisation and represents the super class for IOIDS-SENDER and IOIDS-SOURCE; in fact, it maintains the link to the G4DS entity information, namely the G4DS member identifiers

IOIDS-SENDER information about the sender of an IOIDS event

IOIDS-SOURCE information about the initial source of an IOIDS event

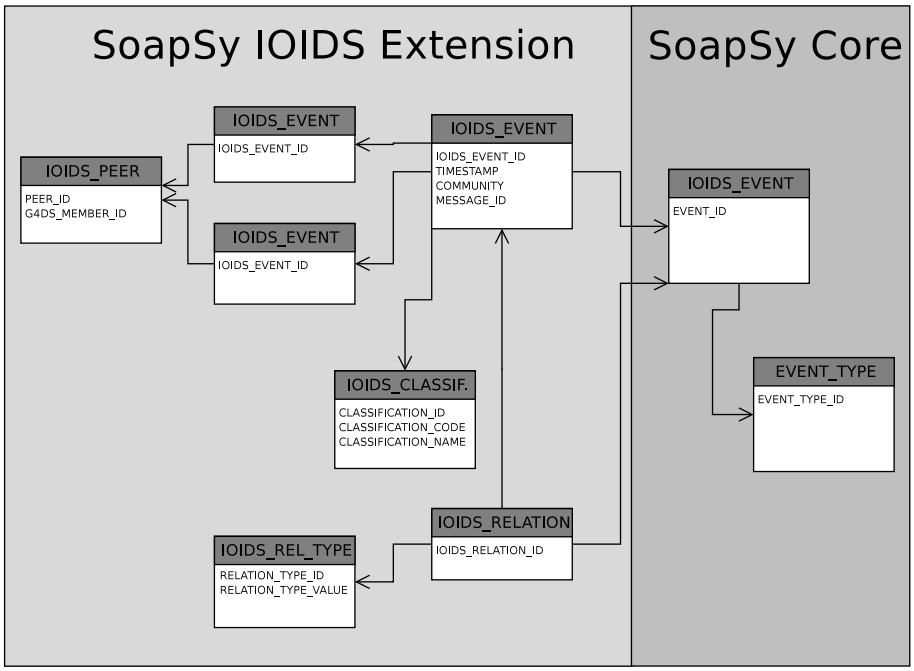


Figure 6.4.: SoapSy IOIDS Extension DB Schema

IOIDS-RELATION-TYPE information about types of relations between events (e.g. derived events, or parent events - for more information check section 6.3.2)

IOIDS-RELATION for maintaining relations between one IOIDS event and none, one or more further SoapSy events

IOIDS-EVENT the major entity for an IOIDS event; maintains the link to the actual SoapSy (core) event and maintains links to the other relations of the IOIDS extension

On the resources CD the complete collection of additional relations including their attributes employed for storing IOIDS extension information in XML representation is shown. The syntax of this file is aligned to the XML database approach and is explained in greater detail in section 6.2.3.

Integration with IOIDS and modules for Extensions The same way, IOIDS is making use of SoapSy extension facilities other event generators are doing so (check section 7.3.3 for an example using SnortDB event data). Consequently, for each event inserted into the SoapSy database there is always exactly one entry in the event table in the SoapSy Core and exactly one entry in one extension of the SoapSy extension part. IOIDS itself knows about the layout of the SoapSy core and of course about its own extension. This way, it is able to carry core

information (normalised information, each event generator has to provide) through the IOIDS network together with IOIDS status information. For detailed information, however, for each of the other extensions (such as Syslog, Snort, Windows Event Log) it needs to understand their language, to say, the database layout of their extension.

For these reasons, a plug-in mechanism has been put into place. Referring to figure 6.1 on page 132 these ones are connected to the IOIDS core using a plug-in manager. Each of them is responsible for the following tasks for its corresponding SoapSy extension:

- Collect additional extension information for an event on request of the data engine
- Generate XML encoded information from the available extension information to be sent to other IOIDS nodes as part of the IOIDS event messages
- Parse the incoming extension information and process it into the internal data structure for this extension
- Write extension information into the local database

The plug-in manager is able to determine the corresponding extension (and, consequently, the plug-in) by evaluating the attribute EVENT-TYPE within the Core event relation. The link between the extension event and the core event, however, has to be maintained within the extension event relation. (For example the IOIDS extension relation IOIDS-EVENT maintains a foreign key constraint to the core event table with an attribute named *event-id*.)

The modular approach of handling the extensions enables IOIDS to exchange events from other subsystems via IOIDS channels. As soon as a plug-in is available on the sender's side, the extension information will be included in the IOIDS event message; if, additionally, a plug-in is available for the extension on the receivers side, it is able to integrate this information with the local data repository. However, if the plug-in is missing on either or both sides, the core information can still be exchanged between all nodes. Some further issues of integrating data from third party event generators are discussed further down in this chapter in the implementation part in section 6.3.2.

Connect to database via SOAP RPC

By now, the type of the database management system and the database design have been discusses in very detail. Not yet addressed was the mechanism to connect to the database and access its information. It was stated before that IOIDS is making use of an XML based database as a backend data repository. The access to this repository is gained through a light-weight remote procedure call (RPC) interface, which has as well been developed within

the Information Security Group of the University. It is working with technologies such as XML and the Simple Object Access Protocol (SOAP) (Box et al. (2000)) and provides a very simple, but nonetheless, extremely powerful remote interface for accessing the database.

Features of *XSM* and its way of working are discussed in detail in (Xynos (2005)); however, the following list gives a brief overview:

- XSM listens on a TCP port for incoming SOAP connections
- Every request to XSM must be encoded in XML
- XSM separates database requests (SELECT) and database modifications (updates / inserts)
- Results for the database transaction in the backend are returned in XML encoding as well; in the case of a modification this would be the primary key for the top-level event, in case of a request in contrast, the result set is returned in XML encoding
- Complex data may be inserted using a specific XML structure aligned to the constraints between the relations
- Basic implementations for fetching complex data are in place; implemented using joints
- XSM needs to know about the database design it is working on; consequently, an XML encoded descriptions of all relations with their attributes and constraints has to be provided (as compensation, the corresponding SQL statements for creating the entities within database can be derived from it using XSM tools)
- XSM may be configured to be used with any relational database (including type conversion in both directions)

In the current stage, the information about the database design has to be provided in a single file; consequently, database information for any SoapSy extension to be integrated has to be added to this file. (XSM is still under development and the distribution of database information across several files is thought to be covered in future developments). However, the complete copy of the XSM file for the database design of SoapSy with extension for IOIDS and SnortDB can be found on the resources CD.

Fetch latest events

In the overview for IOIDS (section 6.2.1) the need for IOIDS to be aware of the arrival of new events in the core event database has been mentioned already. The mechanism handling this issue should satisfy the following requirements:

- The notification of IOIDS should be in near-realtime ("only" near-realtime because the entire IOIDS topology cannot work in realtime anyways due to constraints showing up when using the internet such as latencies and the like)
- Events must not get lost; even if the system is down for a certain time or must be rebooted for some reason
- The fetching of data should be handled efficiently; hence, even on occasions of high data volume to be reported, the traffic caused by the database communication should be kept as small as possible.
- The process must be performed accurately and events should arrive in order of their arrival in the database.

When examining the attributes of the database four different approaches become obvious for performing this task, namely:

- Evaluation of the database internal object ID (OID)
- Evaluation of the timestamp of the events
- Evaluation of the event identifiers of the events
- Trigger in conjunction with a stored procedure on the database, finally invoking a function on the (local) observing system

The former three approaches have many things in common, whereby the latter one follows a totally different strategy to achieve the goal. These ones are:

- The action is always initiated on the database client side
- Some kind of status information must be maintained on the client side (the lastest timestamp of the latest id)
- The value for the attribute (either one of the IDs or the timestamp) for event n must be greater than the one for event n-1

One major advantage of this kind of approach is its simplicity of implementation. In frequent time intervals, the IOIDS trigger is performing a request to the database and asks for all entries, whose id or timestamp is greater than a certain value. Another advantage here is the ability to fetch several events at the same time, and with the option to change the frequency of the fetch cycle, it becomes very scalable. Finally, by making the status

information about the latest id or latest timestamp persistent somewhere in the local file system, no data would get lost whenever the system is going down for some reason; meaning, all events arriving in the meantime can be fetched with a delay. Last but not least, it would be simple to support a group of IOIDS triggers, since each of them would maintain its own status information. The major shortcoming of this approach is first of all a design issue: although, trigger functionality is provided on the database, it is evaded and implemented using a "software trigger". Furthermore, there would be overhead on the network for checking for new events, whenever no new events have arrived since the last check.

The attributes of the datebase trigger implementation are almost opposite to the ones aforementioned for the software trigger. First of all, the db trigger looks to be a sober implementation of the given problem. Furthermore, it would tackle the issue of near-realtime best, since the function would be invoked by the trigger right after the event has been inserted. The major shortcoming for the datebase trigger is its complexity of employment, which is getting even worse due to the utilisation of the XML based data base approach. Literally, this trigger would need to bypass the XML db interface, which would interfer with a clear, modular software architecture. Although, not causing any overhead when no event is arriving in the database, the trigger implementation will fire every time for each event, which might cause problems when a high number of events arrive in a relatively short time frame at the database. Last but not least, it will be difficult to register several instances of IOIDS with the trigger or define a trigger for each of them.

Since the realtime issue is not pushed too far and the potential break of the design of the overall architecture weighs heavier than the soberness of a single component, the decision has been made for the software trigger. In particular the event id has been chosen for the problem due to the following (potential) problems with the other two attributes:

- An event arriving at the central SoapSy database might be equipped with a timestamp smaller than the one for an event, which is in the database already due to differences in time on several connected nodes or the like. After all, the timestamp attribute appeared not to be an ideal candidate for a reliable identifier for new events within the database relations.

- Although the database object identifiers are known to be increasing all the time, this behaviour is depending on the implementation of the database management system itself. The event id, however, is declared in the database design as *serial*, which ensures the requirement for the attribute being always greater for event n than for event n-1.

Finally, the features and performance issues of the implementation chosen are examined in the experiment part and described in part of section 7.3.2.

6.2.4. Message Formats

This section is dealing with the encoding of information, exchanged between the IOIDS nodes on the network. First of all, as employed on many other occasions within the project, an XML based approach has been chosen due to the well-known and widely explained reasons.

Afterwards, several well-known formats for exchanging computer incident related information have been evaluated against their use for exchanging IOIDS data. The outcome of these evaluations are basically, that it is possible to squeeze IOIDS data into well-known formats such as IDMEF (Debar et al. (2005)) or IODEF (Demchenko (2003)); however, the result would be that the source of information squeezes as much knowledge as possible in the known fields of the standard protocols and puts all the remaining information in self-defined extensions of the protocol. The receiver(s) of the message would take all this information, only concatenate it in a way to bring it back into the initial database aligned layout (see section 6.3.4 for details). At the location of employment of the message format, it is not intended to be accessed by any software but IOIDS itself. These reasons overweighed the always desired aim to incorporate well-known standards and, finally, an IOIDS specific XML based message format has been put into place, which is pretty much aligned to the internal data structure of IOIDS, an this way, to the backend database. The following sections explain this dataformat in detail; and furthermore provide information about options to integrate knowledge from the aforementioned third party event generators.

In the objectives (section 3.3.4) the requirement for two different types of messages has been discussed in detail; namely the knowledge requests and information updates. Each of the following two sections is explaining one type in detail.

Information updates

Information updates are generated by the data engine, whenever the local node decides to inform other nodes of the IOIDS network about a certain state of the data repository. (This behaviour depends on the policies in place and is discussed in more detail further down in this chapter in section 6.3.2.)

The IOIDS information update message must always start with a top-level *IOIDS* tag. An attribute of this node with the name *type* must be set to value *update* in order to identify the IOIDS message as an information update. (This distinction is also done before using action strings on the interface from the IOIDS application to G4DS; however, it ensures that messages have been dispatched appropriately. See section 6.3.1) The remaining part of the IOIDS message is split into two parts:

- The major IOIDS event

- None, one or several related events

A complete IOIDS information update message including one related event of extension type *snortdb* is provided on the resources CD. An overview of the final structure is shown in listing 6.1 (attributes are indicated with a leading asterix):

Listing 6.1: Structure of IOIDS information update request message

```
ioids
  * type='update'
  ioidsevent
    * all attributes of ioids event such as timestamp, community, ...
    event (Core event for the major IOIDS event with all its attributes and relations)
    remaining IOIDS extension relations such as sender, source, classification
  relations
    relation 1
      * type='parent' (indicates the relation type to the ioids major event)
      plainevent (plain event without any extension information in case the receiver
          does not understand the extension)
        event (Core event for the related event with all its attributes and relations)
      extension
        * type='extension_name' (name of the extension such as SnortDB, Syslog or
            WinEventLog)
        extension specific information for the related event
    relation 2
      ...
    relation 3
      ...
    ...
    relation n
```

As shown in the listing, there must only be a single major IOIDS event, but a number of related events may be attached to it. The following two paragraphs provide some more detailed information about the two parts of an IOIDS update message.

Major IOIDS event The major IOIDS event carries all information available in the database for the actual event to be populated. Its structure is completely aligned to the internal IOIDS data structure and therefore to the SoapSy database layout for IOIDS extensions. (see section 6.3.4). This way, it can be assembled easily using a generic XML wrapper and parsed on the receiver's side using a generic parser / information extractor. Besides the information about the IOIDS extension, of course it also needs to include the information about the corresponding SoapSy core event, which is carried inside the IOIDS extension event message.

Related events As previously stated, none, one or more related events may be carried inside one IOIDS information update message. They have to be wrapped by a *relations* tag, the order of the related events does not matter. Each relation has to be equipped with a *type* attribute, which indicates the type of relation between the major IOIDS event and the current related event. Inside each of the relations, the event is carried twice, ones including the extension information and ones without extension information; hence, the plain SoapSy core event. The plain event is compulsory, the extension event in contrast is optional. (This is for the reason that the sender might not support the extension of the given event itself or does not want to provide all the details regarding its employed policies.) The plain event again contains all attributes and relations gained from the SoapSy core for this event. The extension event itself contains all the information from its corresponding extension within the SoapSy database, but should nevertheless contain all the core information as well; however, this is left to the implementation of the plug-in for this extension. (section 6.2.3 and 6.3.2)

There is no reply for information updates. If any node requires more information about this event or some kind of related knowledge, it has to generate a knowledge request itself. (see following section 6.2.4)

Knowledge requests

Knowledge request messages are generated whenever the local node (in fact the data engine as part of the IOIDS core system) needs more information about a certain event, events from a certain timeframe or maybe events involving parties from a certain address range. In fact, the group of events requested may be defined using the following parameters:

- Events occurring within a certain timeframe
- Events, where one or several of the involved agents (source, destination, observer or reporter) hits certain address information such as a certain IP address range or a DNS name
- Events of a certain classification
- Events, which were reported by a specific extension (evaluation of the event type)

For implementing IOIDS knowledge requests, two messages are required, namely the knowledge request query and the knowledge request reply. The knowledge request query defines the aforementioned parameters, whereby the knowledge request reply carries the set of events, suiting the given requirements. Both them are encoded in XML and have the *ioids* tag as

root element. In detail, each of them is explained with their structure in one of the following sections.

Knowledge request query As implemented for the information update, the knowledge request root element (*ioids*) must be equipped with an attribute called *type*. In case of an knowledge request query, it must be set to *request-query*. Afterwards, the conditions for the query are included in XML encoding.

For each condition to be taken into account, the following information must be transmitted:

- The name of the attribute
- The value for this attribute in the data format suiting the aforementioned attribute
- An operator, saying, how to compare the value of the request with the data in the data repository.

Consequently, each condition entry looks the as described in listing 6.2:

Listing 6.2: Format of an IOIDS information request query condition

```
<condition attribute="attribute_name" operator="operator_name">VALUE</condition>

e.g.
...
<condition attribute="source_ip" operator="equals">192.168.0.5/255.255.255.0</
    condition>
...
```

The format of the value depends on the given attribute. In case of an DNS name, for example, this should be a string. Whenever an IP address is expected, however, a couple of IP address / netmask (optional) should be provided. For a date / time specific attribute the well-known format for timestamps must be provided.

The operators are the same for each of the attributes. Currently supported operators are:

EQUAL equals

NOTEQUAL - unequal

GREATER - greater than

SMALLER - smaller than

Finally, the various conditions must be brought into context; hence, the logical relationship between them has to be mirrored within the message. The only allowed relationships between conditions are:

- logical *AND*
- logical *OR*

In order to support nesting of conditions with different relationships (e.g. one node wants to request all events for a certain IP address range as source of the event but occurring on two different timeframes), a grouping concept has been introduced. Each group is equipped with a relationship identifier (either *OR* or *AND*). Each item within a group might either be a condition (as described above) or another group with is own relationship identifier. (see also the example given in listing 6.3)

Using the explained facilities, a request query could be created, which shall cover all events attacking IP addresses in either of range *192.168.0.0-192.168.1.255* or of range *192.168.10.0-192.168.10.255* in the time frame covered by December 2005. The result for this query is shown in listing 6.3.

Listing 6.3: Structure of example of an IOIDS information request query condition

```
ioids
   * type="request-query"
   conditions
      group
         * relationship = "AND"
         condition
            * attribute="event.timestamp"
            * operator="GREATER"
            2005-12-01 00:00:00.00000
         condition
            * attribute="event.timestamp"
            * operator="SMALLER"
            2005-12-31 23:59:59.99999
         group
            * relationship="OR"
            condition
               * attribute="event.destination_ip"
               * operator="EQUAL"
               192.168.0.0/255.255.254.0
            condition
               * attribute="event.destination_ip"
               * operator="EQUAL"
               192.168.10.0/255.255.255.0
```

Knowledge request reply After the incoming IOIDS knowledge message request has been processed inside the IOIDS data engine, the result set of events (hitting the given filter criterion) are assembled into a knowledge request reply and sent back to the sender of the initial knowledge request query.

As employed for all IOIDS messages, the root element of the XML encoded messages must be assigned the tag *IOIDS*. The compulsory attribute for this element named *type* must be set to the value *request-reply*. Inside this root element follows a single instance of the element *results*, which indicates the beginning of the result block. This element is equipped with an attribute *count*, which carries the total number of events sent in this message. Each event is wrapped inside a *result* element and is provided either in one or two representations; namely the SoapSy core event and the extension event. (The concept has been introduced earlier on in this section when talking about information updates) Consequently, the listing of the plain event is compulsory, the extension event information, however, is optional.

Listing 6.4 visualises the structure of an IOIDS knowledge request reply message. Again, attributes of nodes are shown with a leading asterix.

Listing 6.4: Structure of IOIDS knowledge request reply message

```
ioids
  * type='request-reply'
  results
    * count
    result
      plainevent  (plain event without any extension information in case the receiver
          does not understand the extension)
        event (Core event for the event with all its attributes and relations)
      extension
        * type='extension_name'  (name of the extension such as SnortDB, Syslog or
            WinEventLog)
        extension specific information for the event
    result 2
      ...
    result 3
      ...
    ...
    result n
```

All information provided within such a message must be protected appropriately. These issues, however, are not part of the message format discussion, instead, they are discussed further down in this chapter in section 6.3.2.

6.2.5. Security Policy

The requirement of protecting the knowledge of the IOIDS system adequately has been outworked as a major issue in the objectives (section 3.3.4). Many efforts have been put on researching into the definition and application of security models - a selection of very popular representatives has been introduced in section 2.5.3.

As one representative, the *Chinese Wall Security Policy* (Pfleeger and Pfleeger (2003)) has been identified as a very popular and widely implemented security policy. Due to its protective approach, its support for communities and its highly common acceptance it has been chosen as the base for the security policy for IOIDS. However, there exist some problems with the Chinese wall security policy, which requires slight modifications for its application to IOIDS, namely:

- The dynamic environment does not suit our requirements at all (the decisions about access to objects is based on previous accesses to other / same objects). For IOIDS, however, the status of objects (events) are not dynamic at all - ones they have been assigned a community and a classification, they will not change anymore.

The application of a status for each object would have been to heavyweight. Instead, an object may exist in different communities (domains) at the same time by simply inserting another copy of the object. Each of these may then be equipped with a different level of protection, saying its classification.

Trust

Trust has been identified as a major issue for the implementation of mechanisms to exchange this sensitive security related information. Trust can only be established between two parties; the technology in its nature is only a tool, which has to align to the out-laid trusting guidelines and restrictions; hence, it is in charge to implement sufficient technologies which are able to support the requested policies.

IOIDS supports these requirements by an implementation of communities and classifications, working out the following way:

- *Communities* have been introduced in G4DS already. They form a group of parties, which maintain certain trust levels between each other. These communities are mapped into the IOIDS system; literally, each piece of information within IOIDS (which should normally be made up by an event) must always belong to exactly one community. It is only valid for this particular community and must not escape this distribution domain.

Whenever an event shall be distributed throughout several communities, one IOIDS message has to be created for each of them.

- *Classifications* are introduced as a novel approach in IOIDS layer. Each piece of information within IOIDS must be equipped with a classification identifier. Classifications are unique throughout the IOIDS system and range from private (id 0 - only for the local node) to public (id 10 - information accessable for all members within all communities). This allows the maintenance of information of private nature as well as public nature within the same data repository. The classifications are discussed in detail in the following paragraph.

The community of a message as well as the classification are stored directly together with the actual event information within the central database as part of the IOIDS event. (see also IOIDS database layout in section 6.2.3) This tight relation between the information makes up the base of protecting the knowledge of the local node.

In practise, each piece of information is tagged with the following information when stored on a node:

- Origin of the message (includes both, the actual sending node of the message and the Trusting Community it was created in)
- A classification for the message itself, which provides information about the ways this message is supposed to be used.

The origin may be determined easily by processing header information of the message which has been sent before. (However, a sending node might be member of several TCs, this way it is required to define into which TC the current message is sent.) The classification for the message must be defined by the sending node. (For more details about available classifications for messages also check section 6.2.5) Consequently, the following formalism is introduced:

$$k = \{m, s, t, c\}$$

These symbols represent the following information; a piece of Knowledge (k) is made up by the combination of:

- The message itself (m)
- A source for this message (s)

- A Trusting Community of this message (t)
- A classification for the message (c) defining, how to handle, process and distribute it

If a newly created knowledge chunk k_x is considered, all its related information is represented the following way:

$$K_x = \{M_x, S_x, T_x, C_x\} \rightarrow k_x$$

with:

- $M_x = \{m_1, m_2, ..., m_a\}$ with a as number of messages,
- $S_x = \{s_1, s_2, ..., s_b\}$ with b as number of sources,
- $T_x = \{t_1, t_2, ..., t_c\}$ with c as number of TCs involved and
- $C_x = \{c_1, c_2, ..., c_d\}$ with d as number of classifications.

Simplified, Figure 6.5 visualises the model employed for managing knowledge within the IOIDS architecture. It does not mirror the entire complexity of the model since the relations between the knowledge chunks have not been provided with the figure.

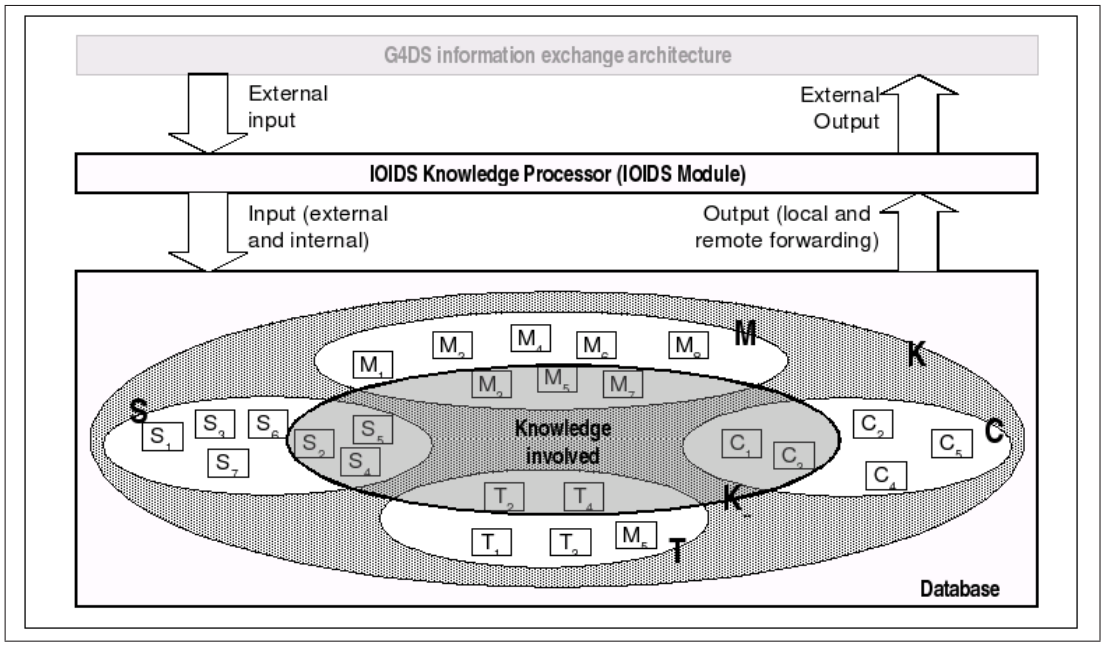


Figure 6.5.: IOIDS Knowledge Management

A knowledge pool (K) is made up by the sets of messages (M), Sources (S), Trusting Communities (T) and their classifications (C). Pieces of information in the knowledge pool have relations between each other. In fact, no item in the knowledge pool may exist multiply; hence, each message is put into relation with its corresponding sender, source Trusting Community and Classification.

Whenever a new message is being created, a certain subset of pieces of knowledge is involved. The overall subset is named K_x with all its members (or sub-subsets) M_x, S_x, T_x and C_x for the messages, sources, trusting communities and classifications involved for creating this new message. In an example as described in Figure 6.5 three source messages ($M_x = \{M_3, M_5, M_7\}$) with their 3 source addresses ($S_x = \{S_2, S_5, S_4\}$) from two different communities ($T_x = \{T_2, T_4\}$) and classifications ($C_x = \{C_1, C_3\}$) have been involved. Finally, the entirety of the knowledge subset K_x results in the new piece of knowledge k_x. The new piece of knowledge k_x comes as a unit of message, source node, destination community and classification; hence, at the same time new entries will be asserted into the data repository. (At least one new entry for the message has to be inserted; the values for classification, Trusting Community and Source Node may be existent in the database already.)

By storing the information about the source, the trusting community and the classification of a message together with the message itself, it can be made sure that knowledge never escapes from its supposed distribution domain.

Concept of Sanitising

In section 3.3.4 within the objectives it has been made clear, that in order to protect an organisation's information appropriately without preventing too much knowledge from being exchanged the garbling of knowledge is a widely discussed approach. Often, this identity related information is not even required at the receivers side for recycle the information; for instance does the existence of the source IP address for a snort event not impact the processing at the receiver's side as long as they may put corresponding information into context.

IOIDS integrates features for garbling in its conceptual design; the implementation on network communication level however, does not yet support this kind of feature. (See also ideas for future work in section 8.3) Basically, the principal aim to protect the identity of the origin of a piece of knowledge is coming in two stages:

- Prevent the receiver or any involved party from gaining knowledge about the identity by gathering and evaluating network traffic
- Do not enable anybody reading the presented information to backtrace knowledge within the message to the origin. (For example it is very likely that event genera-

tors such as intrusion detection systems put address information about the attacked victim into the generated event.)

The design is based on these two stages too. The implementation of the former aim has been named *anonymising* and must take place on G4DS layer. The latter one is named *sanitising* and has to be carried out on application layer, saying, within IOIDS itself. As previously stated, the idea only exists in its conceptual design - it has been discussed in great detail in the publication (Pilgermann and Blyth (2004, ISBN: 0-9547096-2-4)). For now it was only to make clear the necessity of dedicated classification levels for supporting sanitising.

Finally, sanitising may only be performed by the origin of the message itself. No other node is able or allowed to perform sanitising on behalf of the source node. Even, if no sanitised version if available (or not yet available), but sanitised versions are the only ones to be forwarded, the actual (un-changed) message must then not be used to create the derived message.

Determine destination community

Before the classification of a message can be calculated its destination Trusting Community has to be appointed. Goal is the highest possible propagation of the knowledge without violating any restrictions made up by the classifications rules of all the messages in the knowledge pool K_x for the new message.

Source Trusting Communities as well as classifications of input messages have to be examined. (Sanitising bits of the classification rules do not need to be taken into account for this procedure since it is assumed that the Trusting Community of a message would be garbled in the process of sanitising). The following rules are applied to the knowledge pool in exactly the given order in order to determine the destination Trusting Community of the new knowledge chunk k_x.

1. There is at least one message in the knowledge pool, whose classification is specified as C0 - Local confidential (including its derived grades for sanitising C1, C2 and C3). The new message will not have any destination Trusting Community. (This message (or any derived message) must not be send to any party; hence, the applying of a Trusting Community is superfluous.)

$$\exists\, x \in C_x \,|\, x \in \{C0, C1, C2, C3\} \longrightarrow t_x = n.a.$$

2. If there are at least two messages with classification C_4 - Destination Confidential (or one of its derived grades for sanitising C_5 and C_6) being originated in two different Trusting Communities, then there won't be any destination TC assigned for k_x.

$$\exists\,(c_1,c_2,t_1,t_2)\in(C_X,C_X,T_X,T_X)\,|\,c_1,c_2\in\{C_4,C_5,C_6\}\,\&\,t_1\rightarrow c_1\,\&\,t_2\rightarrow c_2\,\&\,t_1\neq t_2\;\longrightarrow\;t_x=n.a.$$

3. There are at least two messages with classification C7 or C8 - Community Boundary Protected whose source TCs are different. – No destination TC will be assigned.

$$\exists\,(c_1,c_2,t_1,t_2)\in(C_X,C_X,T_X,T_X)\,|\,c_1,c_2\in\{C_7,C_8\}\,\&\,t_1\rightarrow c_1\,\&\,t_2\rightarrow c_2\,\&\,t_1\neq t_2\;\longrightarrow\;t_x=n.a.$$

4. There is at least one pair of messages with classification C4 (or C5, C6) for the first one and classification C7 (or C8) for the second one, which are originated in different Trusting Communities. – No destination TC may be applied.

$$\exists\,(c_1,c_2,t_1,t_2)\in(C_X,C_X,T_X,T_X)\,|\,c_1\in\{C_4,C_5,C_6\}\,\&\,c_1\in\{C_7,C_8\}\,\&\,t_1\rightarrow c_1\,\&\,t_2\rightarrow c_2\,\&\,t_1\neq t_2\;\longrightarrow\;t_x=n.a.$$

5. All messages in the knowledge pool are originated in the same Trusting Community. – The destination Trusting Community equals the one of the source messages.

$$t=\alpha\,\forall\,t\in T_x\;\longrightarrow\;t_x=\alpha$$

6. Messages from different Trusting Communities are involved; however, only exactly one message is marked with classification C4 - Destination Confidential (or one of its derived grades C5 and C6). – The destination Trusting Community is the one of the message in relation with trusting community C4.

$$\exists\,(c,t_1,t_2)\in(C_x,T_x,T_x)\,|\,c\in\{C_4,C_5,C_6\}\,\&\,t_1\rightarrow c\,\&\,t_1\neq t_2\;\longrightarrow\;t_x=t_1$$

7. All messages with protection C7 / C8 - Community Boundary are originated in the same Trusting Community.

$$t=\alpha\,\forall\,(t,c)\in(T_x,C_x)\,|\,t\rightarrow c\,\&\,c\in\{C7,C8\}\;\longrightarrow\;t_x=\alpha$$

8. All messages are public. – The destination TC is the one with the most entries for processed messages with this Trusting Community.

$$c=C10\,\forall\,c\in C_x\;\longrightarrow\;t_x=\alpha\,|\,\alpha=\max(m_x,t_x)\,\&\,m_x\rightarrow t_x$$

Determine classification

In order to determine the classification for a new piece of knowledge two situations have to be considered:

- A single message has been processed or several messages have been processed in order to create the piece of knowledge; but all the messages are with the same tags and from the same source.
- Several messages have been processed in order to create the piece of knowledge, processed messages come from different sources (nodes or communities) and have different tags applied.

The former case is easy to handle; however, the latter one requires more efforts in order to align to distribution policies. The following rules are applied to determine a classification for a new message regarding to the sources and classifications of all messages being used for creating this new message (the rules in here are ordered, this way the first rule applicable for the knowledge pool of the new message K_x will be used and the processing is terminated).

Overview classifications Classifications for messages are ordered, a classification with a higher number represents a lower protection than one with a low number; hence, the classification 0 stands for the highest protection of the message ever - meaning, only the creator itself is using and processing the data.

ID 0 – Private – Knowledge is exclusive for the local node. Even nodes with total trust will not be able to access information in here.

ID 1 –Local Confidential, but Sanitised for Third Party The actual message needs to be protected completely, however, sanitised messages may be sent to destination party.

ID 2 – Local Confidential, but Sanitised for Community

ID 3 – Local Confidential, but Sanitised for Everybody

ID 4 – Destination Confidential Information from this message may only be used and processed on the receiver's node itself. Neither the message nor any message which is (partially) derived from this message may be sent to any other party.

ID 5 – Destination Confidential, but Sanitised to Community

ID 6 – Destination Confidential, but Sanitised to Everybody

ID 7 – Community Boundary Protected This message may be circulated throughout the source Trusting Community. Any derived message may be handled the same way; however, neither the message itself nor any (partially) derived message may ever leave the community boundaries.

ID 8 – Community Boundary, but Sanitised to Everybody

ID 9 unused

ID 10 – Public – Knowledge may be read by any node.

Determination in two stages The classification determination is performed by two progressive stages. In the first stage sanitising options will be left behind and only the major class (c_{tmp}) will be calculated. Four major classes are available; namely $C_{tmp}A$ - Local Confidential, $C_{tmp}B$ - Destination Confidential, $C_{tmp}C$ - Community Protected and $C_{tmp}D$ - Public. The second stage takes into account all the sanitising information and will this way calculate the final destination classification.

1. At least one message of the knowledge pool is classified Local confidential C0 or Local confidential with any of the Sanitising options (C1, C2 and C3). – The temporary classification of the new message is $C_{tmp}A$ - Local confidential.

$$\exists\, x \in C_x \,|\, x \in \{C0, C1, C2, C3\} \;\longrightarrow\; c_{tmp} = C_{tmp}A$$

2. At least one message of the knowledge pool is classified Destination Confidential C4 or Destination Confidential with any of the Sanitising options (C5 and C6). – The temporary classification of the new message is $C_{tmp}A$ - Local confidential.

$$\exists\, x \in C_x \,|\, x \in \{C5, C6, C7\} \;\longrightarrow\; c_{tmp} = C_{tmp}A$$

3. All messages in the knowledge pool, which are classified as C7 - Community Protected or its Sanitised classification C8, are originated in the same Trusting Community as the Destination Community of the message. – The temporary classification of the new message is $C_{tmp}C$ – Community Protected.

$$t = t_x \,\forall\, (t, c) \in (C_x, T_x) \,|\, t \rightarrow c \,\&\, c \in \{C7, C8\} \;\longrightarrow\; c_{tmp} = C_{tmp}C$$

4. There is at least one message classified as C7 Community Protected or its sanitised classification C8 which is not originated in the destination community of the message. – The temporary classification of the new message is $C_{tmp}A$ - Local confidential.

$$\exists\,(t,c) \in (C_x, T_x)\;|\,c \in \{C7, C8\}\;\&\,t \neq t_x\;\longrightarrow\;c_{tmp} = C_{tmp}A$$

5. All messages in K_x are classified public C10. – The temporary classification of the new message is public $C_{tmp}D$.

$$c = C10\,\forall\,c \in C_x\;\longrightarrow\;c_{tmp} = C_{tmp}D$$

After determining the major destination class, the final destination classification is calculated by applying the following rules in the given order:

1. The temporary classification is Local Confidential $C_{tmp}A$. There is at least one source message with classification C0 - Local Confidential. – The destination classification is C0 - Local Confidential.

$$\exists\,c \in C_x\,|\,c = C0\,\&\,c_{tmp} = C_{tmp}A\;\longrightarrow\;c_x = C0$$

2. The temporary classification is Local Confidential $C_{tmp}A$. There is at least one source message with classification C1 - Local Confidential but Sanitised for Third Party or C4 - Destination Confidential. – The destination classification is C1 - Local Confidential but Sanitised for Third Party.

$$\exists\,c \in C_x\,|\,c \in \{C1, C4\}\;\&\,c_{tmp} = C_{tmp}A\;\longrightarrow\;c_x = C1$$

3. The temporary classification is Local Confidential $C_{tmp}A$. There is at least one source message with classification C2 - Local Confidential but Sanitised for Community or with classification C5 - Destination Confidential but Sanitised for Community or with classification C7 - Community Boundary protected. – The destination classification is C2 - Local Confidential but Sanitised for Community.

$$\exists\,c \in C_x\,|\,c \in \{C2, C5, C7\}\;\&\,c_{tmp} = C_{tmp}A\;\longrightarrow\;c_x = C2$$

4. The temporary classification is Local Confidential $C_{tmp}A$. – The destination classification is C3 - Local Confidential but Sanitised to everybody.

$$c_{tmp} = C_{tmp}A \longrightarrow c_x = C3$$

5. The temporary classification is Destination Confidential $C_{tmp}B$. There is at least one source message with classification C4 - Destination Confidential. – The destination classification is C4 - Destination Confidential.

$$\exists\, c \in C_x \,|\, c = C4 \,\&\, c_{tmp} = C_{tmp}B \longrightarrow c_x = C4$$

6. The temporary classification is Local Confidential $C_{tmp}B$. There is at least one source message with classification C5 - Destination Confidential but Sanitised for Community or C7 - Community Boundary Protected. – The destination classification is C5 - Destination Confidential but Sanitised for Community.

$$\exists\, c \in C_x \,|\, c \in \{C5, C7\} \,\&\, c_{tmp} = C_{tmp}B \longrightarrow c_x = C5$$

7. The temporary classification is Destination Confidential $C_{tmp}B$. – The destination classification is C6 - Destination Confidential but Sanitised to everybody.

$$c_{tmp} = C_{tmp}B \longrightarrow c_x = C6$$

8. The temporary classification is Community Protected $C_{tmp}C$. There is at least one source message with classification C7 - Community Boundary Protected. – The destination classification is C7 - Community Boundary Protected.

$$\exists\, c \in C_x \,|\, c = C7 \,\&\, c_{tmp} = C_{tmp}C \longrightarrow c_x = C7$$

9. The temporary classification is Community Protected $C_{tmp}C$. – The destination classification is C8 - Community Boundary Protected but Sanitised to everybody.

$$c_{tmp} = C_{tmp}C \longrightarrow c_x = C8$$

10. The temporary classification is Public $C_{tmp}D$. – The destination classification is C10 - Public.

$$c_{tmp} = C_{tmp}D \longrightarrow c_x = C10$$

Destination confidential messages are not created during this process; however, this classification may be assigned whenever a reply is send to any node in reaction to a knowledge enquiry. Another occasion is considered with the existence of some piece of information, which is only related to a certain node and it's essential to inform this node; however, this node shall not be enabled to pass further this information nor any message derived from it, not even within the community. The classification "Destination Confidential" provides an opportunity to the application to publish information exactly this way.

6.3. Technical design and implementation

6.3.1. IOIDS as a service - Integration with G4DS

This section discusses in detail all measures for and the interaction of the Inter-Organisation Intrusion Detection System with the subjacent Grid system (G4DS) from the view of the IOIDS. The integration of applications from the view of the grid system itself has been discussed in very detail in section 5.3.6.

In the grid explanations, it has already been worked out that for each application or so-called *G4DS service* a XML formatted service description has to be provided. A copy of the service description for the IOIDS service is provided on the resources CD.

Any service, which intends to make use of the facilities provided by G4DS needs to authenticate and register in the first place. This process is explained in the first of the following three sections. The only way for a grid service to interact with G4DS is the sending and receiving of messages through it. All required meta data (community, distribution domain, such as single member or a group of members, or other protective delivery options, such as sanitising invocation) is transmitted during these processes. The passing and retrieving of information is discussed in detail in the following section. Afterwards, attention is drawn to the determination, transmission and extraction of the classifying attributes for messages, namely the classification itself and the trusting community, its belongs to.

Register service and establish connection

In the introduction for service integration within the G4DS chapter (section 5.3.6) the procedure of integrating a service with the grid platform has been discussed already. From the view of the connected service, the IOIDS service in our case, the following steps have to be applied (assuming, the grid infrastructure with a working community and members, which know each other, is in place already):

Service description (KSDL) – As part of each G4DS service, a service description (XML) must be developed. The service description for IOIDS is provided on the resources CD. This one has to be made available to the G4DS system using its maintenance interface (section 5.3.3).

Access control extensions – Furthermore, any G4DS service may make use of the access control facilities, provided by G4DS (G4DS access control in sections 5.2.4 and 5.3.4). For these reasons, new G4DS access control policy files have to be developed and made available to G4DS by placing them at a certain location within the local file system and registering them using G4DS configuration files.

Service Key – Whenever a grid service intends to make use of G4DS facilities, it has to authenticate at the G4DS system with the private key of an asymmetric key pair at connection time. (section 5.3.6). This key must be created within the G4DS maintenance environment and can be exported from there. The G4DS service client module provides an API for connecting against G4DS using this private key.

Connect – Ones, the private key has been exported, it will be passed on to the G4DS platform with a rendezvous request. This methodology is implemented by the G4DS service client module and finally only requires the service id of the connecting service and the private key. (a generic example is provided in listing 5.9 in section 5.3.6)

Passing and retrieving information

Most information for the IOIDS system is encapsulated inside the IOIDS message. Some information, however, is essential for delivering the message to the distribution domain. Consequently, along the actual messages passed to the grid system, some meta data has to be passed on as well, which is made up by the following items:

- The distribution domain of the message, which might either be a single member (specified by the member id), or a group of members (specified by community id or service id).
- The community, this message is valid for and shall be delivered in
- The actionstring of the IOIDS system, indicating which action shall be performed on the receiver's side.

The definition of community is optional; in fact, if no community is specified, G4DS attempts to determine the best suitable community for the given combination of sender, receiver(s) and service.

The action string has to be passed on to G4DS in order to allow incorporation of G4DS access control for IOIDS purposes. In fact, by providing policy files for the G4DS access control system, connected applications can make use of it, and this way, messages may be filtered even before they are passed on to the application. Finally, it is left to the application to check, whether the given action string really suits the data transmitted with the message. (see also section 6.3.3 for more details)

6.3.2. Dataengine - process information and react appropriately

The IOIDS data engine has been identified as an substantial module within the IOIDS architecture in the design introduction in section 6.2.1 already. In fact, whenever any data is to be shifted within IOIDS the data engine is involved; no matter at which location the data is originated and no matter which nature the data is of.

In order to support the demand of a highly configurable approach, the IOIDS data engine has been based on a policy driven approach. This way, it is directly connected to a policy repository, which accesses backend information in XML format from the file system. More information about policies and their processing is provided in the following sub section.

Based on the policies, the data engine may initiate any (or a combination) of the following actions with the new information:

- Create a new event for the locate data repository
- Distribute the information throughout the IOIDS network
- Request more information regarding certain attributes

Each of these items may occur several items within the chain of actions; so that, for example, two different events may be generated for distribution into two different distribution domains such as communties.

The latter two items of the list draw the attention back to the already discussed separation of message types; namely the knowledge requests and information updates (section 6.2.4 in message formats). This clear separation has to be aligned to since this is the way to make the difference between read and write access to the IOIDS network.

Besides the issues, which may be covered with the policy repository the data engine is in charge to handle two problems, arising for the IOIDS:

- The overall IOIDS network is thought to be made up by a number of different trusting communities, which will, by any chance, exchange information between each other. In certain constellations of communities with their gateways between each other, there

might occur circles, messages could travel through. The data engine must make sure that a message, which has been processed ones already, is not processed another time.

- The SoapSy database with its separation in Core and Extensibles has been discussed in section 6.2.3. The integration of data from the core is rather straight forward due to its static nature. The extensibles, in contrast, are rather dynamic, which requires a modular approach within IOIDS to support the processing of their data.

Each of them is discussed briefly in a dedicated sub section further down in this document.

Policy approach

The Inter-Organisation Intrusion Detection System approach is supposed to be highly configurable to allow implementation in very different employment scenarios. As visualised in Figure 6.1 in the introduction of this chapter, the data engine is connected to a policy repository, which maintains its data on the local file system.

In general, the policy files are XML encoded instructions on the local file system, which define exactly, how the data engine has to react in response to certain events. The following types of information is present in the policy descriptions:

- Input information, describing the situation / event a certain action shall be triggered. The corresponding parameters are:
  - Source of event (local or remote)
  - Content of certain fields within the event (e.g. event source IP)
  - Pattern, which may occur anywhere within the event description (fulltext search)
  - Timeframes for the occurrence of the event
  - Sub-System (Extension) of event (only for local event)
  - Trusting Community, the event was delivered in (only for remote event)
  - Origin (G4DS member id) of the event (only for remote event)
  - Classification of event (only for remote event)
- Reaction information, indicating, which action shall be carried out. As briefly stated before, the list of actions is made up by:
  - Create a new event for the locate data repository (Required parameters: Classification, Trusting Community)

- Distribute the information throughout the IOIDS network (Required parameters: distribution domain (single user, certain community / communities, whole IOIDS network))
- Request more information regarding certain attributes (Required parameters: enquiry domain (single user, certain community / communities, whole IOIDS network), enquiry attributes (see section 6.2.4 for details))

Note that in order to carry out action 2, action number 1 has to be performed before. No event is to be populated into the IOIDS network without existing inside the local IOIDS data repository.

An extract of a sample IOIDS policy as shown in listing 6.5 makes the idea of policies more understandable. It defines a policy rule, which passes any message from any of the local subsystems to all members within the two communities C001 and C002. (A full copy of the default IOIDS policy is provided on the resources CD.)

Listing 6.5: Structure of extract of IOIDS data engine policy

```
ioids-policy
   rule
      id = 00033
      situation
         origin = local
         subsystem = *

      reactions
         reaction
            * number="1"
            type = NewLocalEvent
            parameters
               classification = 7
               community = C001
               distribute
                  domain = C001
                     * type="community"
         reaction
            * number="2"
            type = NewLocalEvent
            parameters
               classification = 7
               community = C002
               distribute
                  domain = C002
                     * type="community"

         reaction
```

```
        * number="5"
        type = Terminate
```

A policy is made up by an ordered list of rules. As indicated, each rule within the policy is equipped with a unique identifier. This is very important since the processing of rules is carried out in the order given by these IDs.

Besides the identifier, each rule is coming with two further sections:

situation - Describing parameters, for which this rule is valid for.

reactions - Formulating the actions to be triggered in response to the certain situation.

The parameters in the situation environment do not need to be given in a certain order. Supported keywords in there are:

origin Side of origin for the event. May only be *local* or *remote*.

subsystem Name of the SoapSy subsystem, which inserted the given event. (A comma separated list allows the specification of several subsystems in here.)

timeframe Date and time, this event reaches the data engine. Specified for date and time in form of tags *starttime* and *endtime*. (This is not necessarily equivalent with the timestamp of an event since there might be latencies on the way, the event is passed from the actual source of the event (SoapSy event observer) towards the IOIDS system.)

classification Specifies a certain classification for an event. (A range may be given by providing a list of comma separated values.) This may only be applied to remote (IOIDS) events - new local events are not yet equipped with a classification; the assignment of one is part of this process.

community The given event is valid for a certain community. (A list may be supplied by separating values with commas.) As stated for the classification, this item may only be used for remote events (origin *remote*).

sender The sender of an IOIDS event (defined by its G4DS member id). (A list may be supplied using comma separated values.) Again only valid for remote events since local events do not yet have an IOIDS sender.

event.source The source of the actual (SoapSy core) event. A variety of information may be supplied here, such as IP address (net mask for address ranges), host name, protocols, ports or operating systems.

event.destination The destination of the actual (SoapSy core) event.

event.observer The observer of the actual (SoapSy core) event.

event.reporter The reporter of the actual (SoapSy core) event.

event.timestamp The timestamp of the actual (SoapSy core) event.

The structure for the four items *event.source*, *event.destination*, *event.observer* and *event.reporter* are the same, following this guide:

- If supplied, either an IP address or a (fully qualified) hostname should be given.
- For IP addresses, a netmask may be supplied in order to cover a certain range of IP addresses.
- For network intrusion detection events (as for example generated by snort) the knowledge about the network protocols is of high interest. The protocol here may be defined for layer 4 of the TCP/IP protocol stack (saying, most likely TCP or UDP).
- The port may be provided optionally. Depending of the nature of the information, either the source or the destination port is considered.
- Information about operating systems may be supplied on a *start-with* basis; the IOIDS data-engine performs a string pattern search and checks for matches between the operating system of the event and the string provided here.

The format of the data provided within the descriptions tags depends on the tag itself. A classification, for example, is to be given as a simple string; an event.source, in contrast, is made up by a structure integrating information about the IP address, DNS name, operating system etc..

The reactions have to be defined for execution in a certain order, indicated by their reaction number. This takes the very likely requirement into account, to carry one certain reaction before or after another one. Reactions are categorised; the supported reaction types are:

NewLocalEvent – Generate a new event for the local SoapSy event repository. As stated in Listing 6.5, distribution of this event may be defined as part of this rule with a distribute parameter and more information defining the distribution domain. Further parameters have to be provided for assigning an IOIDS classification and a community for the new event. (By providing the value *Auto* for classification or community, the policy process as described in section 6.2.5 is kicking in and attempts to determine the requested values by processing information.)

Terminate – Abort processing of reactions with this rule.

Refer – Continue processing of reactions with rule with the number to be given as a parameter.

Each of them may or has to be equipped with a certain number of parameters. Refer to the full example on the resources CD for getting detailed information about their handling. With reaction *refer* the processing will be continued at the rule with the given number and come back to this hierarchy after finishing the reaction rule tree. Reaction *terminate*, in contrast, aborts the processing for the policy immediately.

Control flow of data

The network of G4DS nodes, and consequently of IOIDS nodes, is structured in communities. Defined by the G4DS community descriptions, nodes may be member of several communities at the same time and even pass on information from one to another. This behaviour has been introduced with the concept of *Trusting Community Gateways* in section 5.3.5. The definition of trusting community gateways, however, only provides information about the permission of passing of messages; not in detail, which messages are to be passed on. This, in fact, is left to the service running on top of G4DS. It has to provide information within its distribution policies, which defines this behaviour. This leaves it finally to IOIDS to take care of the issues of distribution, and this way, problems arising by distributing throughout several communities, such as:

- Circles in information flow
- Bouncing of messages between two parties

Since IOIDS follows a peer-based approach, there is no central instance deciding about the distribution of messages. Despite, the population of a single message depends on the behaviour and configuration of several parties:

- The actual initiator of the message, which has to define a distribution domain and needs to flag the message with a classification to provide guidance for further processing
- Certain gateways on the route towards the final destinations must decide, whether they should pass on a certain message, drop it or only process it locally
- The final destination itself, which decides whether the received information is to be processed or dropped

Each of the involved parties bases its decisions on their policies (see previous subsection 6.3.2). Due to the dependence for the population of information on so many parameters, a sufficient way has to be put into place in order to avoid messages bouncing within the G4DS / IOIDS network. The solution for overcoming this problem has been the introduction of message identifiers.

Message Identifiers In order to avoid multiple processing of the same information, each event has to be equipped with a local message identifier (*mid*). This mid is directly stored together with the event information in the SoapSy IOIDS extension relation named *IOIDS-EVENT*. Furthermore, a list of ids for messages is maintained, where the messages have not been processed by the local node (due to policy decisions). This prevents the re-processing of messages against IOIDS policies.

Whenever a new event is created on the local node, a new message id has to be produced and stored together with the event. This is then passed with the IOIDS event in order to allow receivers to check for previous occasions of the message. Message IDs have the following attributes:

- They are unique throughout the entire IOIDS network. This is achieved by combining information from three different sources for the message id, namely the member identifier, a timestamp and a random part.

- Each version of an event has a different identifier; meaning, two identical messages for different communities possess different ids as well as the sanitised version of a message is equipped with a different id than its original message.

Integrate data from third party event generators

IOIDS is intended to be used as a knowledge exchange mechanism on a high level of communication between several parties integrating a variety event logging applications, such as:

- (Distributed) Intrusion Detection Systems such as *Snort*, *SnortNet*, *Prelude* or any other Intrusion Detection system product

- System event loggers such as *syslog* or *Windows Event Log*

- Any other application contributing information in the structure of the SoapSy event core

It has been stated in section 6.2.3 already, that integration of event loggers is performed using the approach of a central data repository called SoapSy. However, the integration of third-party information for IOIDS is a two step process:

1. The third party event generator must be able to contribute data to the SoapSy database and provide the information in the required format
2. IOIDS must be able to pick up this information and distribute it to the IOIDS network

Step number one is actually beyond the scope of the IOIDS architecture itself; however, for experiment and analysis purposes some efforts have been put on the development of such event data converters. Some outcome is presented in form of Snort converting tools within the analysis chapter in section 7.3.3.

Step number two, however, is directly part of the requirements to be addressed by the IOIDS system (section 3.3.4). The way, IOIDS is dealing with this requirement is the utilisation of modules, one for each extension, and the employment of an *ExtensionHandler*, which acts as kind of a dispatcher for retrieving and inserting extension information for any extension. There are only four use cases the extension modules are required for, which are explained in detail.

Use Case I - Insert a new Event into SoapSy – Whenever a situation requests the storing of an event into the SoapSy data repository (triggered by the dataengine - section 6.3.2), there might be extension data attached to the event, which is desirable to be stored together with the plain event information. (Section 6.2.4 explains in detail, how several events with extension information may be attached to a single IOIDS event.) With the separation of core and extensible within SoapSy (and consequently the way IOIDS is accessing it), IOIDS is always able to store the core event information of the events; for the extension information, however, it passes on the request to the extension handler, which attempts to load the corresponding extension and passes on the request thereupon.

Use Case II - Retrieve event information from SoapSy – The retrieving of information pictures the same problem: whenever a related event shall be loaded from SoapSy, IOIDS itself may only request the core event information. The extension specific information however, cannot be retrieved - simply due to the lack of knowledge about the database layout of its relations.

Use Case III - Send a new G4DS message containing extension data – Besides the SoapSy database the data engine is also connected to the grid system G4DS. All messages con-

taining extension information must be formatted appropriately as well. The XML encoding of the extension bits for the events has to be performed by some kind of module, which knows about the layout of the corresponding extension.

Use Case IV - Receive a new G4DS message containing extension information – The reverse direction for the G4DS messages, namely the processing of extension data in XML format and transformation into the IOIDS internal dictionary based data structure can only be performed by a subsystem, which knows about the structure, too.

Consequently, for each extension there has to be a module maintained for IOIDS, knowing about the database structure of the relations for the SoapSy extension of the corresponding extension. Whenever there is a request from IOIDS, it is in charge to transform data from the SoapSy XML interface layout into the internal IOIDS dictionary layout (section 6.3.4) and vice verse. So far, the following modules are maintained for IOIDS:

SnortDB – Module, which processes knowledge as provided by the SnortDB tool, explained in section 7.3.3

IOIDS – For the reason that one major IOIDS event may refer to other IOIDS event, the actual SoapSy IOIDS extention information of the linked event should be processed too. Therefore, an IOIDS extension module had to be introduced.

The list of extensions is configurable. Ones, a new module for a new SoapSy extension has been completed, it may be registered for IOIDS use within the IOIDS configuration module (config.py) and will be picked up by the extension handler immediately after restart.

The implementation of a module for each extension to be supported requires quite a lot of efforts and it is essential, that these modules are kept up to date - meaning, whenever there is a change in any of the SoapSy extensions, the IOIDS extension module has to be updated at the same time; otherwise, only core event information will be carried for this event type. A dynamic approach, which is capable of processing the information automatically is highly desirable. In the section for possible future extensions (section 8.3) an idea is mentioned, which parses SoapSy database description information and configures itself dynamically for all available SoapSy extensions.

6.3.3. Access Control

In the design section for IOIDS the theoretical approach for protecting information within IOIDS has been introduced and explained broadly (section 6.2.5). This section, in addition, will explain, how the aforementioned measures are carried out in practise.

As employed for the access control mechanisms for G4DS (section 5.2.4 and 5.3.4 for the implementation), the access control for IOIDS is making use of the following objects:

- Actors
- Targets
- Operations

Practically, an actor is performing a certain operation on a certain target. The target within IOIDS is always the destination node itself (saying, the receiver of the message); hence, it is not so much of interest due to the lack of changing. The actor is considered to be the original sender of a message (meaning, if a message has been routed through G4DS, it is the original member rather then the last gateway on the route). The member (actor) is expressed by the use of its G4DS member id. The operation is expressed in form of an action string again, for now the following actions are supported:

ioids.write.newevent – The new incoming message for the IOIDS service is an IOIDS knowledge update and attempts to insert at least one new event into the IOIDS knowledge repository.

ioids.read.requestinfo – The new incoming message for the IOIDS service is an IOIDS knowledge request and, this way, requests more information about events stored in the local IOIDS knowledge repository.

The data engine is in charge to apply the correct action string for a certain message to be sent through G4DS. The action string for IOIDS is additionally passed on to the G4DS system together with the actual message, which allows integration of IOIDS access control with the G4DS access control mechanisms for a prefiltering before incoming messages are actually passed on to the IOIDS system (see following section). Further examination of actions and reactions is carried out by the data engine itself using the supplied policy repository (see section 6.3.2 for details).

Integration with G4DS access control mechanism

For the subjacent grid topology G4DS a rule-chain based access control mechanism is introduced in section 5.3.4. Any connected application may make use of these facilities by passing on action strings together with the actual message. This way, messages may be filtered by the G4DS access control mechanisms before they are actually passed on to the IOIDS system.

Additional G4DS access control policies (section 5.2.4) have to be provided and integrated in order to facilitate the process.

Ones, the required access control policies have been provided for G4DS, the whole process of access control for IOIDS is performed using the following steps:

1. The data engine in IOIDS becomes aware of the need of populating a new message into the G4DS network. Regarding its location of processing it determines the appropriate action string for the message request to be carried out. The new message is passed on together with the action string and the distribution domain to the *G4DS Connector*.

2. The G4DS Connector passes the message on to the connected G4DS system including the meta data about distribution domain and action string.

3. The grid system assembles a new G4DS message, which contains the actual (IOIDS) message as payload, the service id for the IOIDS service and the action string - both the latter in the G4DS message meta data. This message is distributed to the members as indicated by the distribution domain received from the IOIDS G4DS Connector.

4. On the receiver's side, the G4DS message is parsed and the payload and the meta data information is extracted. After determining the destination service, the meta data information is handed over to the G4DS access control for checking against the rules given in the access control policy files.

5. In case of a positive reply from the access control, the application, which has connected against G4DS under the given G4DS service id, is contacted and the incoming message is passed including meta data information such as action string, community and sender of the message.

6. The IOIDS G4DS Connector receives the message and passes it on to the IOIDS data engine as an incoming remote message including the meta data information.

7. The IOIDS data engine thereupon parses the payload and performs checks on the available information, whether the action string suits the given payload data. In case of a successful outcome of this check, the given event is processed against the IOIDS policy rules (6.3.2) and defined actions will be carried out by the data engine.

For a complete copy of the IOIDS rules for the G4DS access control mechanism check the resources on the CD.

6.3.4. Internal Datastructure

IOIDS is dealing with plenty of structured data, such as information about events with their related information about involved parties, payload data and the like as well as information from several extensions for events. A way has been to be found, which supports the internal handling of data and makes converting into the required (mostly XML encoded) data formats easy.

The approach employed for IOIDS is based on a mixture of datatypes dictionaries and lists and supports greatly the maintenance of relations between several structures. Furthermore, the wrapping from and into the XML based formats may almost be done in a generic way. The following (simplified) example (given in listing 6.6) for an event with some related information explains the structure in some more detail:

Listing 6.6: Example for data structure within IOIDS

```
['event',
  {('timestmp', '2005-12-12 10:53:39.46339'),('event_id', '23')},
  [ ['event_type',
      {('event_type_id', '2'),('event_type_name', 'ioids')},
      []
    ],
    ['source',
      {('src_name','Unknown Source')},
      [ #... agent and other relations here ...# ]
    ]
  ]
]
```

In the example given in listing 6.6, lists are indicated using square brackets, dictionaries in contrast are indicated using cambered brackets; each item inside a dictionary as a couple in round brackets as (key, value).

In fact, each structure is made up by an ordered list of exactly three items, which represent the following information:

1. The name of the structure (in String representation)
2. The attribute of the structure (in dictionary representation)
3. All related structures for this structure (in form of an unordered list)

The dictionary of attributes for a structure (the second item within the structure list) holds all attributes for it, the key is taken by the name of the attribute, the dictionary value by the attribute value respectively.

The items in the relation list (this list is the third item within the major structure list) holds a list of structures itself; hence, each item of this list is exactly of the same layout as the one just explained.

The content

The content of the structures is supposed to hold computer incident event information. As stated in section 6.2.3, research in this area has been extensively undertaken, and with SoapSy an approach has been proposed, which integrates event information from a variety of event generating applications. Since IOIDS is supposed to exchange exactly this kind of knowledge, it could take advantage of the thereby achieved results and, consequently, the IOIDS internal data structure more or less mirrors a lists / dictionaries representation of the SoapSy database design.

The changes, IOIDS employed towards the original data layout are covered with the following list:

- IOIDS totally ignores the abstract part of the SoapSy database
- Some indexes have been taken off due to problems of their employment in practise (see top of copy on the resources CD (history) for all details)
- The relation process has been referenced from relation agent instead of relation computer; this enables the event processor to distinguish between two different processes on exactly the same machine at the same time (for example, one acting as the destination (Apache) and another one as observer (syslog))

The relations for the IOIDS extension for SoapSy have been discussed in section 6.2.3. The internal data structure of them is an exact copy, just in lists / dictionaries representation. The actual event within the SoapSy core is referenced thereby from within the IOIDS-event structure. Consequently, the complete IOIDS event is represented the following way inside IOIDS (see listing 6.7):

Listing 6.7: Internal representation of IOIDS events

```
['ioids_event',
  {('timestamp', 'VALUE'),('ioids_event_id', 'VALUE'), ('community_id', VALUE)},
  [ ['event',
      { #...all event attributes...# },
      [ #...all event relations...# ]
    ],
    ['ioids_source',
```

```
      {('ioids_peer_id',VALUE)},
      [ ['ioids_peer',
          {('ioids_peer_id', VALUE), ('peer_memberid', VALUE)},
          []
        ]
      ]
    ]
    ['ioids_sender',
      {('ioids_sender_id',VALUE)},
      [ ['ioids_peer',
          {('ioids_peer_id', VALUE), ('peer_memberid', VALUE)},
          []
        ]
      ]
    ]
    ['ioids_classification',
      {('classification_id', VALUE), ('classification_code',VALUE), ('
          classification_name', VALUE)},
      []
    ]
  ]
]
```

The relations between IOIDS events and other SoapSy core events are mapped using the same structure. The copy on the resources CD shows the complete data structure of the IOIDS extension in XML representation.

Cooperation with database access

One major issue for the decision about the internal data structure was the ability to transform it into the various XML formats. In fact, two different XML formats must be created and parsed from or into the aforementioned structure, namely:

- The XML format for IOIDS messages (see section 6.2.4)
- The XML format used by the XML database

Considering the internal datastructure described in the example in listing 6.6, the corresponding XML output for the former one (the IOIDS message) looks as shown in Listing 6.8.

Listing 6.8: Transformation to IOIDS XML from internal data structure

```
<?xml version="1.0" encoding="UTF-8" ?>
<event timestmp="2005-12-12 10:53:39.46339" event_id="23">
  <event_type event_type_id="2" event_type_name="ioids">
  <source src_name="Unknown Source">
    <!-- agent and other relations here -->
  </source>
</event>
```

It becomes very obvious when comparing the two listings that the transformation from one into the other may be performed very easily, meaning, even with a generic wrapper or parser. This is exactly what IOIDS is doing; however, in order to allow control over the data, methods for certain structures / relations may be defined, which are triggered on their occurrence. These can support certain post or pre-processing activities, such as the conversion for special datatypes or the like.

Finally, the employed data model is absolutely generic; hence, it can be transformed into any required output format. Furthermore, data can be accessed very easily by just browsing through the structure tree using indexes or keys for the dictionaries.

XML / DOM Tree processing and creation

For clear separation of responsibilities, an additional step has been put into place for transformations from the internal datastructure into the XML based dialects and vice-verse. Technologies for XML have been discussed and compared in section 2.6.1 of the state of the art discussion. The *Data Object Model* (DOM) has been chosen for its simplicity and power. This way, IOIDS is never creating any XML code, instead it assembles a DOM tree, which is passed to an XML writer to create the final outcome. This way, IOIDS prevents itself from dealing with any problems occurring within the XML syntax and grammar.

The other way around, IOIDS neither parses any XML document directly, instead it passes the XML encoded string to an XML parser provided by libraries and afterwards processes the information by travelling through the Dom tree.

6.3.5. IOIDS logging facilities

As employed for the G4DS system (section 5.3.8), IOIDS has also been equipped with a logging mechanism. This addresses the execution environment of IOIDS, which is supposed to be in service manner rather than in interactive mode.

IOIDS logging supports the following output targets:

- An IOIDS unique log file in the local file system (in IOIDS syntax)
- The Linux / Unix common *syslog* facilities (Lonvick (2001))

The logging into the local file is performed in an IOIDS specific text-based format, which is, however, derived from the well-known syslog output format. Consequently, each entry is made up by a single line in the file containing a date-time stamp and the log message. An additional action id is included with the entry, which allows easy filtering of messages without processing the actual message string of the log event.

In order to enable IOIDS to log into the system syslog facilities, the local machine has to be prepared for receiving log messages from applications. Afterwards, IOIDS may be set up to make use of these facilities; each log message will be equipped with the identifier *IOIDS*.

Settings for the IOIDS logging mechanism may be applied in the central IOIDS configuration module. Besides the file name of the local IOIDS log file and the indication about the use of syslog facilities, the syslog identifier may be aligned in there.

Logging levels and further settings

IOIDS logging facilities have been encapsulated in a module called *ioidslogging*. A well-defined interface is provided in there for generating new log messages from within IOIDS. In order to allow the user of IOIDS to receive log messages of different detail six logging levels have been introduced. Since each logging message, which is passed on to the IOIDS logging, must be equipped with a log identifier, each of these levels is a set of identifiers, whose corresponding messages are logged in the given level. Thereby, a log level n always includes the log ids of logging level $n - 1$ as well.

For IOIDS the following logging levels are defined:

Level 0 – Critical errors and log system start-up as well as shutdown

Level 1 – Data engine access violations

Level 2 – Incoming and outgoing messages

Level 3 – Message details

Level 4 – SoapSy database activity

Level 5 – All logging events

The access to the logging facilities from within IOIDS is performed with an equivalent interface to the one of G4DS and an example is provided in listing 5.12 in section 5.3.8.

6.4. Conclusion

With IOIDS an approach has been introduced, which enables several organisations to exchange security incident information between each other whilst maintaining the level of trust desired by each of them. The overview of the architecture shows the several components of IOIDS, which are finally hold together by the data engine, in charge for making decisions based on the user defined policies. Thereby, information is processed and populated from and to both, the local data repository SoapSy and remote nodes, connected via G4DS.

Consequently, the major benefits of Inter-Organisation Intrusion Detection, as proposed here, are:

- Computer Incident knowledge may be exchanged in a well-formated, transparent manner.
- The user of IOIDS decides themself, how much they want to trust any other party.
- Employment of access control ensures proper protection of information.
- Utilisation of G4DS as subjacent communication platform ensures a secure, reliable and trustworthy communication.
- Integration of XML-formatted policies allows any user customisation to their needs.
- SoapSy as the central data repository for the local node facilitates the integration of a wide range of event generating applications such as different Intrusion Detection Systems.

It is strongly believed that IOIDS provides a significant improvement for the security of currently employed IT infrastructures. The definition of experiments (chapter 4) lays down the measures for evaluating this statement. In chapter 7 corresponding experiments are documented and a critical analysis of the results supports the expressiveness of this hypothesis.

Chapter 7.

Experiment and Analysis

7.1. Introduction

The general idea of Inter-Organisational Intrusion Detection has been discussed in chapter 3 in great detail. Chapter 4 gave an overview about the design and architecture and defined the way, how to analyse and evaluate them. In the chapters for G4DS (5) and IOIDS (6) more design details and implementation issues have been discussed for the approach. The only thing left for this chapter is the discussion of the results and their evaluation.

The hypothesis has clearly stated that intrusion detection audit data can be exchanged across organisational boundaries in a secure and no-reputable manner. In this chapter the proof of this statement is performed and capabilities of the IOIDS approach are mirrored against the predefined required feature set from the experiment definition and furthermore compared against capabilities of similar or related approaches within the DIDS context.

The analysis is divided into two parts, namely a feature comparison based on evaluation of available documentation and a practical experiment in a laboratory environment. Firstly, the feature comparison is covered with its section; all representatives are mirrored against the predefined feature set and a final overview integrates all available results into tables to allow comparison between approaches. Afterwards, the execution of the practical experiment is discussed in its three stages. Each of them is bringing up its own partial conclusions. In the end an overall conclusion integrates all results from the entire chapter and analyses all available information.

7.2. Comparison of features

As indicated in the experiment definition in section 4.3, this part of the analysis is dealing with a set of approaches in the context of distributed intrusion detection and grids systems and is comparing those ones using a list of pre-defined low-level objectives.

Over the following pages, each representative is dealt with within its separate section, and a small explanation is given for its achievements and short-comings in each attribute category. These is also a list provided for each of them, where the required information was gained from.

In the end of this feature comparison an overview is provided in section 7.2.2, which presents the gained results in matrix form in order to allow easy comparison and drawing conclusions.

7.2.1. Results per representative

Each representative has been introduced in detail in the representative list within the experiment definition (section 4.3.1). In here, only a small introduction with a list of sources is given before the actual feature evaluation is carried out within the given categories.

Inter-Organisational Intrusion Detection System (IOIDS)

Inter-Organisational Intrusion Detection System has been described in much detail within this document. The outcome of this feature comparison is not necessarily restricted to the feature set of IOIDS only. First of all, it is working in correlation with G4DS (as described within this thesis) only. Furthermore, it has been designed and developed with the idea in mind to integrate with an overall solution – consequently, whereever appropriate, features of the overall approach, which IOIDS contributes to, will be included.

Intrusion Detection IOIDS is not coming with a sensor itself. Consequently, all requirements defined for the capturing process have to be addressed by the third-party event generators, logging into the system. As IOIDS is an abstract and integratable approach, any kind of application may be used here - and due to extension cababilities, all their information could be processed and populated.

Correlation for events is down to the employed data engine. For now, only simple integration of information from very different sources and their intelligent persistence preventing any data loss is implemented. However, much research is going on within the group dealing with analysis and correlation of event information and IOIDS would benefit from it directly, as the outcome of those analysis process would go directly back into the database, where it will be picked up by the IOIDS system.

Reporting and response facilities are pretty much down to the analysis equipment connected to the central SoapSy database (which IOIDS is working on as well). Again, IOIDS is not a project, dealing with graphical representation of IOIDS event data and user interaction;

but lots of research within this area is carried out within the research group and IOIDS will benefit from their outcomes directly.

Distribution The distribution process for IOIDS is carried out in a very structured way and allows many configuration options using XML encoded policy files. Consequently, IOIDS stands there as a very scalable and dynamic approach, which is able to map any considerable network infrastructure. This scales from small (fully peer-based) deployment scenarios between a small number of organisations, through hierarchical constellations for a large community up to user-driven communities for private end-users all over the Internet.

The major drawback of IOIDS is the big delays it causes when distributing event information. Consequently, it cannot satisfy the requirement of carrying out the sharing of information in realtime. A list of possible reasons have been named in sections 7.3.2 and 7.3.4. An implementation of an IOIDS topology should however take results from this proof-of-concept implementation and research those problems in more detail in order to speed up the overall process of information sharing.

Security and availability Security has been a major issue throughout the entire process of designing and implementing IOIDS and G4DS. By employing a private-key-infrastructure (PKI) within the communication platform G4DS it can cover loads of the requirements within this category such as authentication, encryption and integrity of information. By further deployment of a policy based access control mechanism within G4DS and its tight integration with IOIDS, authorisation issues could be addressed as well.

Total avoidance of single points of failure has been a major issue for IOIDS and has been pushed in a way that the overall system is completely decentralised and all responsibility for the local node is strictly kept on its side. Credential mapping and single sign-on are not addressed as one can see for up-to-date GRID architectures. However, the attitude is different for a knowledge-grid based system as there is no need to initiate processes on behalf of a certain credential set. Instead, the concept of trusting community gateways makes sure, that information is only passed on between authorised domains.

A major advantage of IOIDS is its modularity in terms of integration of different communication protocols and encryption algorithms. An easy to extend plug-in mechanism with corresponding configuration facilities supports very well these issues.

Last but not least it is capable to resist against DOS attacks in a way to continue work of the overall system if certain components are down, thanks to the peer-based approach. Internal attacks can be ward off by the help of the public key infrastructure and signature validation of every incoming message within G4DS.

Extensibility and collaborative options Due to its component-based approach with integration modules for third-party event generators, IOIDS is downright prepared to interact with and process event information from other Intrusion Detection Systems. All inter-node communication use well-defined standard protocols. It has been developed in Python, which is a cross-platform scripting language. Due to its utilisation of a SoapSy database as central data repository it is well prepared to integrate with other security measures in place as well as a management console based on this platform.

Snort / SnortNet

Snort has been recognised as the de-facto standard for network intrusion detection. As it was noticed, that the SnortNet projects is not much more than an output plug-in for snort and has not gone much beyond a concept and initial implementation efforts, Snort and SnortNet are covered as one approach from here onwards. Documentation has majorly been taken from the Snort website (Snort (2006)) and the only available SnortNet documentation (Fyodor (2000)).

Intrusion Detection Snort is purely a network intrusion detection system and, consequently, limits its detection capabilities to corresponding attack patterns. Advanced features such as stateful inspection can be provided by it. Correlation of event information is performed by the SnortNet component by merging information from different locations together. Reporting facilities are available in form of a web-based frontend called ACID (Analysis Console for Intrusion Databases) (Danyliw (2006)). Although the snort IDS has been reused as capturing application for a number of different projects it does not come with facilities itself to integrate data from other event data generators. Intrusion response and countermeasure activities cannot be carried out using Snort or SnortNet.

Distribution As SnortNet is distributing knowledge directly using a Snort output plug-in, the sharing of information should be performed at least in near-realtime. Due to its fixed structure of sensors, proxies and one management console its is neither very scalable nor very dynamic in the sense of deployment scenarios. Consequently, the only network topology that can be mapped is a hierarchical arrangement of components with one management console at the very top. It is very unlikely that the combination of Snort and SnortNet provides facilities for avoiding duplicate entries in the database as there are problems well-known for the Snort IDS for creating errors alerting exactly this problem of duplicate primary keys.

Security and availability Concerning security issues the combination of Snort and SnortNet is not up-to-date at all. A public key infrastructure and encryption facilities are missing completely (although planned originally), which also ends up in a leek for integrity issues. Authentication and authorisation have been implemented in a very week manner using access lists for source IP addresses on the management console. The missing of a public-key-infrastructure makes the issues of single sign-on and credential mapping obsolete.

As there is a hierarchical arrangement of components employed, one can conclude that the top of the hierarchy, the management console, represents a single point of failure and, consequently, the overall system cannot be DOS resistant in the manner it has been discussed for this project. The checking of source IP addresses of incoming event information cannot provide sufficient protection against internal attacks. The whole Snort application is running as one monolithic application; however, integration of third party intelligence can be introduced by output plugins as well as filters, the so-called Snort preprocessors. Last but not least it is impossible to implement a network using this technology, where each node within that layout maintains its own responsibility.

Extensibility and collaborative options Although, Snort is commonly known as the network intrusion detection system, and, this way, integrated as capturing component in a number of projects (even for the list given here), it does not integrate itself information from third party event generators. The communication between SnortNet components is strictly employed using standards such as the Intrusion Alert Protocol (IAP) and the Intrusion Detection Message Exchange Format (IDMEF). The snort application itself comes as one monolithic application; however a variety of output plugins and preprocessors contribute as modules towards it.

The only way to configure Snort is a configuration file and command line parameters at invocation time. This is not considered to be a straight-forward way to adopt for different deployment scenarios. Although written in the programming language *C*, Snort is available for a number of different UNIX based operating systems as well as for Windows. Due to its vast number of output plugins, which enables it to log into very different output format and output locations, it can be integrated easily with existent security measures. Snort does not directly support integration with known central management consoles (besides its own one named ACID); however, basic communication can be established by sending SNMP traps to management consoles.

Prelude

Prelude is a framework for intrusion detection systems rather than a single intrusion detection system component. It provides components and libraries to establish communication and pass on information and is able to integrate event information from third-party event generators using those libraries. Information has mainly been taken from the on the internet available Prelude Handbook (ThePreludeTeam (2006)) and the somehow outdated Prelude Architecture Guide (Vandoorselaere et al. (2004)). Some experiences from the practical evaluation have flown back into this overview as well.

Intrusion Detection As Prelude IDS has to be considered as a framework it can integrate different types of intrusion detection systems (it even comes with NIDS and HIDS capturing components itself) and is therefore able to cover different detection types as well as it is able to theoretically detect a complete set of attack patterns. This includes attacks spanning over several network patterns, so that they can only be recognised by IDS sensors employing stateful inspection.

Correlation of event information is performed by targeting event information into a single data repository, which may be analysed further at this side. Reporting is supported with the Prelude specific analysis console named *PreWikka*, which comes as a web based front end. As stated in the Prelude documentations, it is also coming with a countermeasure engine, although very limited information is available, what actions can be carried out by it.

Distribution As shown in the experiment, Prelude IDS is well-prepared to distribute event information in real-time. The managers can pass on information to sets of parent-managers, so that a certain amount of scalability is in place; these measures, however, cannot satisfy entirely the objective for deployment in very different and dynamic environments, as there always has to be a top-level manager in the chain event information is travelling. This fact also restricts the Prelude-IDS in its facilities for mapping onto different subjacent network topologies. Last but not least, it could be proved in the experiments that there is no measures in place to avoid duplicates effectively (the user of the program is in charge to not create any circles in the information flow).

Security and availability The category for Prelude-IDS shows very different results for security and availability; as it is showing very good results in the view of security due to its employment of certificates, asymmetric authentication and encryption technologies, but lacking in the view of availability due to its centralised approach.

As information cannot be exchanged in both directions between two links, it is impossible to create a peer-based infrastructure with Prelude-IDS. This ends up in the presence of single points of failure and, consequently, the whole approach is not completely decentralised. Although, fail-over mechanisms are in place and events can be recovered at certain locations within the network the presence of these single points of failure always provide potential for DOS attacks. Internal attacks can be eliminated as far as possible due to the certificates used for authentication.

Further issues arise for the objectives of single sign-on and credential mapping, as any manager that wants to report to parent managers has to create a certificate for each of those managers in question. There is no facilities present for authorisation; meaning as soon as a sensor or child manager has authenticated, it may pass on any kind of message.

Protocols and encryption facilities are hard coded and cannot be adjusted easily to needs for a certain deployment environment. It looks to be impossible to totally control the information on the local node as the node always has to trust the information of ones authenticated members without allowing further checks.

Extensibility and collaborative options As Prelude-IDS is coming as an IDS framework and provides required libraries it is prepared to integrate alert information from third-party event generators. It carries out its communication based on standards such as the Intrusion Detection Message Exchange Format (IDMEF) and Transport Layer Security (TLS).

The core of Prelude-IDS (the Prelude-Manager) comes as a monolithic application and does not spread its functionality over modules. Configuration facilities are in place in order to adjust its behaviour to certain needs; however, the solely setup using configurations files does not support quick changes in an easy manner. Prelude is only provided as source code and it looks like being deployed under UNIX and Linux systems only, although I have not attempted myself to install it on a Windows based operating system.

Locally available security measures can be integrated using the provided libraries only the direction into Prelude. Prelude IDS comes with its own management console PreWikka and is not alerting or reporting into any other management console.

AirCERT

AirCERT is supposed to act as an approach for exchanging high-level event data in an abstract manner between organisations. This puts focus much more on normalisation and integration of data than on low-level capturing processes. In the end, the sensor responsibilities are left to third-party event generators such as the Snort IDS; components for extracting relevant information are supplied.

Information for evaluating AirCERT has mainly been taken from the handbook *AirCERT - The definite guide* (Trammell et al. (2005)) as well as from the project website (Cert/CC (2006)). Some experiences gained during the installation attempts for the practical experiment execution have been used as well.

Intrusion Detection As mentioned, AirCERT benefits from capturing capabilities provided by third-party event generators such as Snort IDS. Consequently, it is able to satisfy the objectives concerning detection types, completeness of detectable attacks patterns and network ids sensor features such as stateful inspection. As major focus is put on passing around information between peers, correlation issues are addressed and corresponding results can be taken further into analysis.

Simple reporting facilities are in place using the so-called *Cheap AirCERT Visualisation Environment* (CAVE), which is, however, very limited in its functionality and cannot be considered as a fully functional management console. AirCERT is not prepared to carry out any intrusion response or countermeasure activities.

Distribution It cannot be made a statement with confidence about the real-time capabilities of the AirCERT distribution process as experiments could not be carried out in the end and results from previously executed experiments or benchmarks by other people are not available. However, the components involved for processing, distributing, receiving and integrating event information (such as a regular expressions processor, a Dav-enabled web-server and a database de-normaliser) let conclude that AirCERT might struggle with this issue.

Concerning the infrastructure bit of the distribution process AirCERT shows its down-side in its complexity and, therefore, cannot be considered as very scalable due to the missing opportunity to deploy in smaller-scale environments. The same issues prevent the user from deploying it to any possible subjacent network topology. Due to a high number of configuration options, it is a rather dynamic approach. Duplicate items within the event information are eliminated using a separate module called *Dedup*.

Security and availability From the security and availability point of view AirCERT benefits from its certificate driven public key infrastructure and its peer-based attitude for information sharing. Consequently, there are measures in place to satisfy the objectives for authentication, encryption and integrity. The certificate approach with its implication that every peer has to get a certificate signed from each other peer it want to communicate with does counteract with the issues of single sign-on and credential mapping.

Theoretically, AirCERT is a purely peer-based approach for exchanging event data. However, certain statements and publications give an insight to the final aim to gather all this kind of information all over the internet and collect it automatically in a CERT database. Although, this approach is implemented in a highly modular way there is no way to simply integrate new encryption algorithms or communication protocols. The aforementioned distributed approach does not provide any surface for DOS attacks due to avoidance of single points of failure as long as not implemented in a way that all event information ends up in a central database.

The decision for the local node is strictly made by the owner of this node only; meaning that only the local node decides about integration and distribution of information. Proper authorisation facilities beyond the separation of information input by sender on the collector‘s side are not in place. These facilities, however, should be able to support the resistance against internal attacks.

Extensibility and collaborative options AirCERT is made to interact with other products within the IDS context and is providing a number of normalisers and integrators to support this job. It incorporates standards such as Secure Hyper Text Transfer Protocol (HTTPS), Web-based Distributed Authoring and Versioning (WebDAV) and the semi-standard Simple Network Management Protocol (SNML) (Center (2003)), which seems to have been created as part of the project.

All its complexity is broken down into atomic components, which facilitates the issues of availability and fail-over mechanisms. Many configurations can be applied to those modules, although this cannot always be carried out in a straight-forward manner. The project's website reports tests with AirCERT on Unix and Linux machines only; claims, however, to be executable on Windows operating systems as well.

Besides the integration of audit data from third party event generators it does not seem to interact a lot with other applications, which might be available as part of the security infrastructure already. On the collector‘s side there do not exist facilities to integrate with potentially available third-party applications for integrated network and security management.

Cooperating Security Managers (CSM)

Cooperating Security Managers is an approach in contrast to the already evaluated ones, which does not provide a platform or framework for exchanging any type of event data; but is rather an approach addressing a particular problem.

Cooperative Security Managers is quite an aged approach already and is discontinued these days. However, the only available source available for this approach is the paper "Cooperating Security Managers: A Peer-Based Intrusion Detection System" (White et al. (1996)).

Intrusion Detection From the view of supported intrusion detection system features CSM seems rather limited in its functionality and is falling behind for features like detection types and completeness of detectable attack patterns. This is, however, a consequence of its attitude to address the particular problem of user tracking throughout networks rather than providing an abstract event information exchange framework. As CSM is not a network intrusion detection system by nature it cannot be measured against the feature of stateful inspection.

Data correlation is a major benefit of this approach and actually builds up the base for its functionality as suspicious behaviour is passed around between nodes and reactions are only invoked ones a certain threshold has been reached over a set of nodes.

Reporting facilities are provided through a CSM specific GUI, which was not available for further inspection to check its feature set. Response mechanisms to certain situations are an essential part of the project and are provided using its dedicated module called *intruder-handling* (IH). As CSM is really focused and narrowed down in its functionality it does not provide options for integrating with any other IDS products.

Distribution It is not really stated in the paper, whether the distribution features are carried out in realtime; however, it looks like that as soon as a certain threshold is met, the distribution is initiated immediately. Although it is a peer-based approach it is not very scalable in the end as every node has to exchange information with every other node in order to track users. The same problems are there for the dynamics of the approach as it is only working if all nodes are exchanging information between each other.

It is not clearly stated how the nodes are made talking to each other; but it does not make the impression that this process is very likely to be mapped onto a variety of different network topologies. The duplicates are not really an issue for CSM, as the decision making process, based on thresholds, does need certain events to occur several times in order to trigger reactions.

Security and availability In the time, CSM was brought up, distributed intrusion detection system were not yet very popular and those early projects did not focus much on security issues for the communication between their components. Consequently, CSM is missing authentication or encryption mechanisms, nor is there anything like a public key infrastructure

available. Certain criterion such as single-sign on or credential mapping become obsolete due to this fact.

Without having keys in place, there is no way to ensure integrity for exchanged information. Limited authorisation features seem to be covered by the module *security manager*, which controls the communication between the local and the remote CSMs. Due to its entirely decentralised architecture, it provides capabilities for DOS attacks as failure of certain nodes do not impact the way, the remaining nodes are working. The responsibility of decision making is entirely kept on the local node.

The implementation of the Cooperating Security Managers has only gone as far as a first prototype, which is not much known about, but some general information about the achieved results. However, it is not shown in the paper that the implementation itself is made up by many components in order to facilitate failover procedures for example. Due to missing authentication facilities it would be impossible for this approach to defend against internal attacks.

Extensibility and collaborative options The CSM approach ended up in a prototype implementation 10 years back and was not concerned about integrating other Intrusion Detection Systems into itself as it was rather a prototype implementation for proving its capabilities for user tracking. Same way, there is no information available that certain standards were employed for communication channels.

As it is very limited in its functionality there was supposingly not much point to equip it with many configuration options. The resulting program was reported to run under Solaris Unix only, although the authors claim it applicability to Windows systems as well. In the end there is neither a way to integrate this approach with potentially available local security measures, nor is there any way reported to interact with a centralised management console.

Autonomous agents for Intrusion Detection (AAFID)

The approach of autonomous agents is a very light weight approach for gathering and reporting intrusion detection audit data within a local area network, as event data finally has to end up in a central data repository called *Monitor*.

This approach is currently not continued anymore; so the lastest information available was in the paper from 1998 named after the approach “An Architecture for Intrusion Detection using Autonomous Agents” (Balasubramaniyan et al. (1998)).

Intrusion Detection As AAFID is an agent based approach, intended to integrate loads of different detection technologies, and even provides the libraries for these purposes it can cover

the list of different detection types as well as it can considered to be capable of detecting the complete set of attack patterns, at least theoretically. This way, network intrusion detection features are finally down to the incorporated agent for this purpose. Correlation of information is a major issue and is covered on several layers within the approach, firstly on the capturing host with a transceiver gathering information from all connected agents; then from monitors, receiving information from several transceivers and in the end a top-level monitor, correlating all information from all available sources.

The latest information available for AAFID stated that no reporting facilities were present, although plans had been created to start working on them. Response mechanisms are not in there in the form of countermeasure engines, but the architecture is prepared to invoke other agents depending on the output of one agent.

Distribution The presented results from the tests with the prototype state that AAFID was not able to carry out the distribution in realtime as the monitor has to wait for results from several transceivers and agents. Due to its approach of monitors and the capability to place them in hierarchies AAFID is considered to be scalable and dynamic in the sense of deployment scenarios. Due to its limitation into hierarchical constellations it cannot be mapped onto the full set of subjacent network topologies. There is not statement made whatsoever about the handling of duplicate event information.

Security and availability The AAFID project team stated that strong and reliable authentication and encryption mechanisms have to be in place. However, as they realised a contradiction with these issues on performance impacts they decided not to include them in the first place. Further discussions are carried on for these issues. Consequently, integrity concerns cannot be satisfied; single sign-on and credential mapping objectives cannot be applied.

In the first place the AAFID architecture makes an impression of being decentralised; however, agents are only autonomous on the host itself; in there they report to a transceiver and are controlled by this as well. Finally, reporting and controlling are left to a central instance, called monitor, which represents a potential single point of failure in the overall architecture and makes it vulnerable to DOS attacks.

In the AAFID paper it is stated itself that their is no access control in place, which ends up in missing authorisation facilities. Together with the lack of strong authentication mechanisms it will be impossible for AAFID to resist internal attacks.

There is not much information available about how much control is left to the transceivers, but it makes the impression, that the (top-level) monitor is in charge to control the connected transceivers underneath it within the hierarchy. Integration of new protocols or encryption

algorithms were not really an issue for the project.

Extensibility and collaborative options Interaction with other intrusion detection systems in only possible unidirectional towards AAFID as it can integrate any kind of application using an agent. There is nothing mentioned within the documents about employing standards for communication channels within the approach. Modularity is definitely the major benefit of AAFID as it is breaking down all its complexity into small agents, where each of them is in charge of a very small set of responsibilities only.

As the experiment execution is not described in detail, there is no information available about configuration of components. The described (second) prototype was implemented in Perl and should run cross-platform. AAFID is not prepared to interact with any potentially available security solutions; nor does it communicate with central management consoles.

Grid Intrusion Detection Architecture (GIDA)

GIDA is the only approach known to date combining the two technologies intrusion detection and grid. Although, the way IDS is applied to grid technology is totally different to the approach of IOIDS, it shows certain similarities for sets of features.

The three papers, which have been published for the Grid Intrusion Detection Architecture (Tolba et al. (2005b), Tolba et al. (2005a) and Tolba et al. (2002)) have served as a information base for evaluation of this approach.

Intrusion Detection The intrusion detection capabilities are hard to measure with the objectives given in here as the attitude of GIDA to protect the grid instead of local networks does implicate the use of totally different sensor and correlation technology. In its view, GIDA does employ different detection types as it is using signature based as well as anomaly based detection mechanisms. A completeness for detectable attacks patterns cannot be reached as it only detects those ones directly related to grid activities. Network intrusion detection is purposefully left out for this approach and, consequently, we cannot apply the issue of stateful inspection.

Communication between sides within the IDS infrastructure is carried out and, this way, correlation of data is performed. So far, results for experiments are only reported from executions with simulations; consequently, there was no real reporting facilities in place. Response mechanisms are not mentioned within the publications at all. Integration with other IDS products is not intended and would not make sense either due to its unique nature for intrusion detection.

Distribution The distribution within GIDA is performed using so-called IDS servers, whereby each IDS sensor may connect and report to several of them in order to provide some fail-over mechanisms. Due to the utilisation of the Globus Toolkit Security Infrastructure (GSI) as one option for event information exchange it is unlikely that information can be distributed in realtime, although no information is provided giving results for it. GIDA claims itself to be scalable and dynamic, which is true in a way that every IDS sensor may connect to different IDS servers; however, there is no information available for opportunities to layout servers in hierarchies in order to prefilter event information and apply the approach to rather large-scale network topologies.

For the same reasons it is difficult to apply GIDA to any considerable network topology as it is rather static in its architecture with sensors and servers. A major advantage of it is the focus on and employment of domains and trust-relationships between nodes. The handling of duplicate information has not been described for this approach and, due to missing experience within the subject area, it is not clear whether this is an issue in general.

Security and availability GIDA is located on top of the Globus Toolkit and makes use of security facilities provided by its security component called Grid Security Infrastructure (GSI). Consequently, it does address issues such as authentication, encryption, integrity and public key infrastructure. There is no information provided about how the IDS servers authenticate between each other; but it may be assumed that the utilisation of the GSI makes them implementing single sign on and credential mapping strategies.

Due to the utilisation of the Globus Toolkit with its information service infrastructure called Grid Information Service (GIS) as subjacent architecture the architecture is fully distributed; hence decentralised and together with the option of choosing several IDS servers for each resource a certain level of DOS resistance can be provided as well. GIDA is a highly modular approach and because of GSI utilisation can take advantage of flexible ways to integrate different communication protocols and encryption algorithms. The provided information let conclude that responsibility is kept on the local node, although the IDS servers put a certain level of control into the network.

Unfortunately, there is no information available about how GIDA is dealing with authorisation issues. Missing this information makes it hard to make any statement about the resistance against internal attacks.

Extensibility and collaborative options The Grid Intrusion Detection System employs a new way of intrusion detection specifically for grids only and does not interact with other third-party intrusion detection systems. It is located around the standards available within

current grid research and is built on top of the Globus Toolkit with its component Grid Information Services (GIS) and Grid Security Infrastructure (GSI).

As GIDA experiments have only been executed on grid simulations no statement can be made about the configuration options of the approach. Globus Toolkit is designed with heterogeneity in mind and, as GIDA is put on top of it, it can directly benefit from that. Integration with local systems can be achieved because it is directly integrated with its host platform Globus Toolkit. There is no information available about how to make it communicate with potential central management facilities.

7.2.2. Overview results and analysis

The following sections provide a structured overview for the just aforementioned results. For each feature category a matrix is presented concluding all the information for the representatives. A small paragraph for each of them discusses the results in some more detail.

Intrusion Detection

Table 7.1 shows the results of comparing the features of each approach with the objectives for the category intrusion detection in detail.

Table 7.1.: Feature comparison: Results category Intrusion Detection

	IOIDS	**Snort**	**Prelude**	**AirCERT**	**CSM**	**AAFID**	**GIDA**
Detect. Types	Yes	No	Yes	Yes	No	Yes	Yes *
Completeness	Yes	No	Yes	Yes	No	Yes	No
Correlation	Yes	Yes	Yes	Yes	Yes	Yes	Yes *
Stateful insp.	Yes	Yes	Yes	Yes	n. a.	Yes	n. a.
Reporting	Yes	Yes	Yes	Yes *	Yes	No	n. a.
Response	Yes	No	Yes *	No	Yes	Yes *	No
Integration	Yes	No	Yes	Yes	No	Yes	No

Only IOIDS and Prelude are able to satisfy the entire set of features within this category. Other approaches are either focussing too much on a certain detection technology and do not allow integration with other products in the context (Snort) or do not provide sufficient facilities for reporting and response.

Distribution

Table 7.2 shows the results of comparing the features of each approach with the objectives for the category distribution in detail.

Table 7.2.: Feature comparison: Results category Distribution

	IOIDS	**Snort**	**Prelude**	**AirCERT**	**CSM**	**AAFID**	**GIDA**
Real time	No	Yes	Yes	No	Yes	No	n. a. *
Scalability	Yes	No	Yes	No	No	Yes	Yes *
Dynamics	Yes	No	No	Yes	No	Yes	Yes
Mapping	Yes	No	No	No	No	No	No
No duplicates	Yes	No	No	Yes	Yes *	n. a.	n. a.
Trust Rel.ships	Yes	No	No	No	No	No	No

None of the selected approaches, not even IOIDS, is able to address all the given issues within this category. As half of the other approaches IOIDS is not capable of carrying out its work in realtime; not even near realtime.

Other approaches are often too static and pre-defined in the way their components have to be deployed; thus, they are not supporting a complete range of deployment scenarios. Last but not least does none of the other approaches support the establishment and processing of trust relationships between parties.

Security and availability

Table 7.3 shows the results of comparing the features of each approach with the objectives for the category security and availability in detail.

Besides IOIDS no approach is addressing the entire set of requirements within the security category. For instance do the aged approaches such as CSM and AAFID not employ any PKI functionality and lack certain features due to this fact.

Further problems with the other approaches are grounded in the employment of central instances for certain responsibilities. This results into susceptibility to denial-of-service attacks. Authorisation is sometimes not provided at all and in some cases to an unsatisfactory extend.

Table 7.3.: Feature comparison: Results category Security and Availability

	IOIDS	Snort	Prelude	AirCERT	CSM	AAFID	GIDA
Authentication	Yes	Yes *	Yes	Yes	No	No *	Yes
Encryption	Yes	No	Yes	Yes	No	No *	Yes
Integrity	Yes	No	Yes	Yes	No	No *	Yes
PKI	Yes	No	Yes	Yes	No	n. a.	Yes
Single sign on	Yes	n. a.	No	No	n. a.	n. a.	Yes *
Decentralised	Yes	No	No	Yes *	Yes	No	Yes
Modularity	Yes	Yes *	No	No	No	No	Yes
Local respon.	Yes	No	No	Yes	Yes	No	Yes *
Credential map.	Yes	n. a.	No	No	n. a.	n. a.	Yes *
No SPOF	Yes	No	No	Yes	Yes	No	Yes *
Authorisation	Yes	Yes *	No	No	Yes *	No	n. a.
DOS resistance	Yes	No	Yes *	Yes	Yes	No	Yes
Int. attack res.	Yes	No	Yes	Yes	No	No	n. a.

Extensibility and collaborative options

Table 7.4 shows the results of comparing the features of each approach with the objectives for the category extensibility and collaborative options in detail.

Table 7.4.: Feature comparison: Results category Extensibility

	IOIDS	Snort	Prelude	AirCERT	CSM	AAFID	GIDA
Int. other IDS	Yes	No	Yes	Yes	No	Yes *	n. a.
Standards	Yes	Yes	Yes	Yes	No	No	Yes
Modularity	Yes	Yes	No	Yes	No	Yes	Yes
Configur.	Yes	No	Yes *	Yes *	No	n. a.	n. a.
Heterogeneity	Yes	Yes	No *	Yes *	No	Yes	Yes
Int. lcl systems	Yes	Yes	Yes	No	No	No	Yes
Int. ctrl. Mgmt.	Yes	No *	No	No	No	No	No

Besides IOIDS no approach can fully satisfy all the issues laid down for this category. As visible in the matrix the major issue for the other approaches is the missing option for integration with a central management console. Besides this one there is not really a trend visible for any of the remaining features and it is more or less depending on the particular

approach in question.

Attention is drawn to the column for the Cooperating Security Managers, which do not address a single issue out of the given list for this category. As explained earlier on in the details for this approach this can be traced back to the age of this approach on the one hand and to the nature of an proof-of-concept implementation for CSM on the other hand.

7.3. Practical experiments

This section explains how the practical part of the experiment was carried out. In section 4.4 of the experiment definition it has been defined what issues have to be addressed with the experiments. In here, the way they have been addressed is described and conclusions from the achieved results are drawn. Technical details for the execution are attached in form of protocols within appendix B - references for details to certain locations within there are provided.

As explained before in the experiment definition, the practical experiments are broken down into three stages, namely:

- Stage 1 – Grid for Digital Security
- Stage 2 – Basic Inter-Organisational Intrusion Detection System
- Stage 3 – Complete Inter-Organisational Intrusion Detection System and comparison

Each of the stages is presented in its own section over the following pages.

7.3.1. Stage 1 - G4DS

The first stage of the experiments is dealing with Grid for Digital Security issues only. Its features are examined by using a simple chat application, which generates small text messages to be passed around. After some information about the setting up of the laboratory environment, the experiment of this stage is presented in the 4 steps, as they were carried out:

1. Populating correct messages
2. Messages population with faulty distribution domains
3. G4DS access control capabilities
4. Penetrating the G4DS network

Results and analysis of each of them is presented in a separate subsection after the information about the laboratory setup.

Installation of laboratory environment

In order to carry out the experiment as described in chapter 4, several nodes had to be set up in a laboratory environment. In this section only the results of this setup will be reported; the complete protocol of the steps are shown in section B.1.1 within the appendixes.

First of all, figure 7.1 describes the employed network topology and its members.

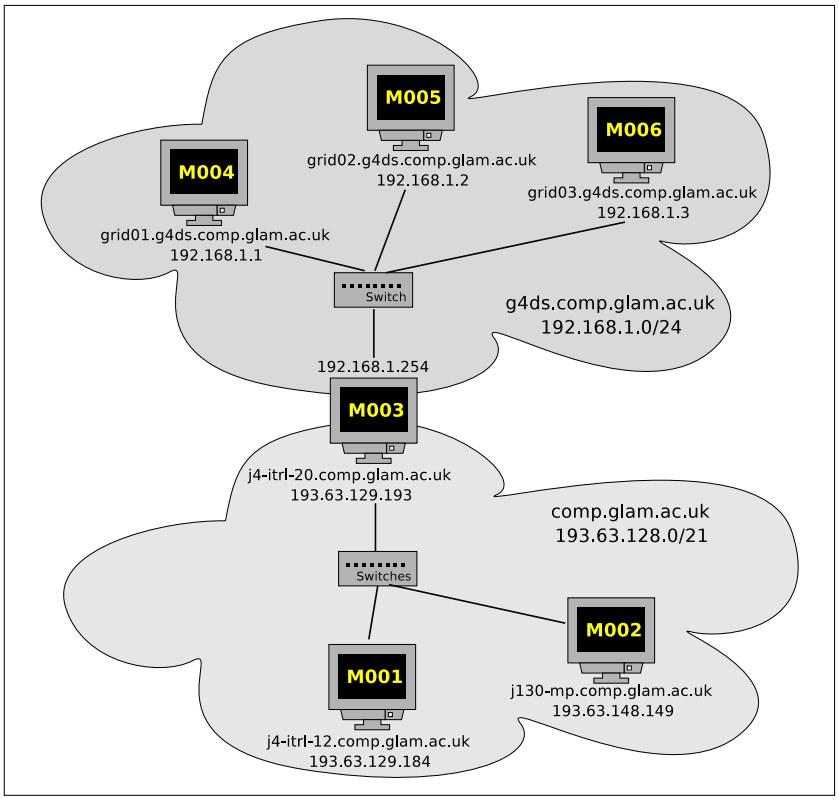


Figure 7.1.: Setup of laboratory environment

The specifications for the shown network nodes are provided in the tables B.7 and B.8 within appendix B.

As mentioned in the experiment definition (chapter 4) and explained in more detail in the architecture description for the Grid for Digital Security (G4DS) infrastructure (chapter 5), each G4DS node must be connected to a database backend in order to make their view of the G4DS network persistent.

The database management system (DBMS) for each node has been deployed on the same machine. (The versions of the are shown in tables B.7 and B.8.) An SQL script is provided for G4DS, which contains all commands in order to set up relations and entities required for it (see listing B.13 for copy). This has been executed against each of the six databases. Table 7.5 provides on overview of the parameters of the databases.

Table 7.5.: Parameters for G4DS databases

	J130	**J4-12**	**J4-20**	**Grid01**	**Grid02**	**Grid03**
Host	j130-mp	j4-itrl-12	j4-itrl-20	grid01	grid02	grid03
Port	5432	5432	5432	5432	5432	5432
User	ug4ds	ug4ds	ug4ds	ug4ds	ug4ds	ug4ds
Password	pwg4ds	pwg4ds	pwg4ds	pwg4ds	pwg4ds	pwg4ds
Database	g4ds	g4ds	g4ds	g4ds	g4ds	g4ds

After preparing the database, the actual G4DS system could be installed. Using the G4DS installation documentation (see listing B.14 within appendixes for copy) this was carried out as described in B.1.1 as part of the experiment protocol. The outcome is a G4DS topology with parameters is described in table 7.6.

Table 7.6.: Parameters for G4DS nodes

	J130	**J4-12**	**J4-20**	**Grid01**	**Grid02**	**Grid03**
MemberID	M002	M001	M003	M004	M005	M006
DefTCs	OK	OK	OK	OK	OK	OK
Name	j130-mp	j4-itrl-12	j4-itrl-20	grid01	grid02	grid03
Logging	5	5	5	5	5	5
syslog	g4ds	g4ds	g4ds	g4ds	g4ds	g4ds
Routing	On	On	On	On	On	On
Policies	OK	OK	OK	OK	OK	OK
DB	On	On	On	On	On	On
Protocols	SOAP	SOAP	SOAP	SOAP	SOAP	SOAP
	TCP	TCP	TCP	TCP	TCP	TCP
Algorithms	RSA	RSA	RSA	RSA	RSA	RSA
	Elgm	Elgm	Elgm	Elgm	Elgm	Elgm

Following the installation and configuration of the G4DS nodes, the communities could be created and applied as defined before in the network topology overview. Going through this procedure, a set of communities has been put into place as described in table 7.7.

After having all the mentioned components in place, the only thing left for this stage was to install and register the chat application.

Table 7.7.: Parameters for G4DS communities

	C001	**C002**	**C003**
Name	g4ds_01	g4ds_02	g4ds_03
Protocols Algorithms	SOAP / tCP RSA Elgamal	SOAP / TCP RSA Elgamal	SOAP / TCP RSA Elgamal
Authorities	M001 M002	M001 M003	M004 M005
Gateways	M001 (C002 in) M001 (C002 out)	M001 (C001 in) M001 (C001 out) M003 (C003 in) M003 (C003 out)	M003 (C002 in) M003 (C002 out)
Additional members			M003 M006

The chat application is a very light weight, text-based chat tool, which makes use of G4DS facilities. Its source code comprises of only a single Python module, which is attached as listing B.17 within the appendix. A G4DS service description had to be created for this application, which is shown in listing B.18. As one can see, the members with ID M001 and M002 act as service authorities for this service.

In order to deploy the chat application, the following steps had to be carried out on each node:

- A private-public key pair had to be created in G4DS in order to allow the new service to connect against the G4DS service.
- The service description had to be applied to the G4DS system.
- The member had to subscribe to the service at one of the service authorities (not for member M001 and M002).

After finishing off these installation steps for the chat application, the created key had to be located in a folder, accessable for the chat application. After starting up the G4DS service, the chat application could be started and was able to connect to G4DS.

The only thing left for the setup was the synchronisation of time between the machines. Details therefore are provided in the experiment protocols in section B.1.1 within the appendix.

Stage 1 - Step 01 – Correct messages

In the first step of stage one messages were distributed from all nodes within the G4DS network to destinations throughout the entire network.

Aim It was examined, whether population of messages is carried out appropriately. Great attention has been drawn to the behaviour of community gateways when requested to pass on a message into another community.

Execution The G4DS service and the chat application are started on all nodes. Afterwards, messages are sent from one node after the using the chat service, which allows specifying the distribution domain on the console when running. After shutting down application and G4DS service, all available logging information and console input / output is backed up.

Results All messages are sent and delivered as expected. Listing 7.1 shows the console output of the chat application on node M001 as an example. Messages, which have to pass through several communities are delivered properly, too - listing 7.2 presents the parts of the G4DS logging output dealing with forwarding a message from community C001 to C002. (The entire logging output of node M001 for this step is presented in listing B.19 within the appendix.)

Listing 7.1: Output of chat application on node M001

```
michael@j4-itrl-12:~/experiment/test chat service$ python testservice.py
Message (q to quit): FR m001 TO m001
Member ID of receiver: M001
Message (q to quit):
        Received (M001 @05/17/06-19:51:03): FR m001 TO m001
FR m001 to m002
Member ID of receiver: M002
Message (q to quit): FR m001 TO m003
Member ID of receiver: M003
Message (q to quit): FR m001 TO m006
Member ID of receiver: M006
Message (q to quit): FR m001 TO c001
Member ID of receiver: C001
Message (q to quit):
        Received (M001 @05/17/06-19:52:26): FR m001 TO c001
FR m001 TO c002
Member ID of receiver: C002
Message (q to quit):
        Received (M001 @05/17/06-19:52:46): FR m001 TO c002
FR m001 TO c003
```

```
Member ID of receiver: C003
Message (q to quit): q
michael@j4-itrl-12:~/experiment/test chat service$
```

Listing 7.2: Partial G4DS logging output on node M001 for experiment stage 1 step 001

```
2006-05-18 11:01:50 199 New incoming message
2006-05-18 11:01:51 198 -- MSG ID Z778370 | SENDER M002
2006-05-18 11:01:51 198 -- Size of msg (brutto | netto): 26822 | 7214 Bytes
2006-05-18 11:01:51 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:01:51 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:01:51 298 Sending control message
2006-05-18 11:01:53 299 New outgoing message - direct delivery (M003 | C002)
2006-05-18 11:01:53 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:01:53 296 -- Size of Data 26938 chars
```

Conclusion Messages within the G4DS network are transported and delivered properly. In detail, the following statements can be made:

- Messages between two nodes within the same community are delivered directly.
- Routing of messages through several communities is working, also with more than one hop.
- Messages, which have a community id as destination, are delivered to all known members of this community.
- Messages are passed on to the correct application regarding the service identifier, the message has been received for.

Stage 1 - Step 02 – Faulty distribution domains

The second step of this stage evaluates behaviour within the network in case of distribution problems.

Aim In contrast to the first step, this step checks response to faulty behaviour. This includes invalid distribution domains as well as partial unavailability of components.

Execution The G4DS service and the chat application are started on all nodes. After applying changes to the network in order to suit one of the following use cases, messages are attempted to send between several nodes:

- The destination for a message represents an invalid identifier (such as an unknown member or community id)
- The chat application is not running on the destination node and, this way, messages cannot be passed on from the G4DS service
- The G4DS service itself is down on the destination node (either on members with gateway functionality or without)

After shutting down application and G4DS service, all available logging information and console input / output is backed up.

Results G4DS has shown the expected behaviour. Unknown destinations have been marked as such within the G4DS logging output (an example is given in listing 7.3, which represents unsuccessful messages deliveries to unknown identifiers). Furthermore, whenever a node acts as a gateway between two communities, shutting down the application does not prevent it from carrying out this job - service messages to this nodes, however, are marked as undelivered within the G4DS logs of the destination node (the corresponding part of the G4DS logs of node M003 are presented in listing 7.4).

Listing 7.3: Partial G4DS logging output on node M002 for experiment stage 1 step 002a

```
2006-05-18 12:19:27 295 Outgoing service message - resolved destination string (M007):
    [u'M007']
2006-05-18 12:19:27 900 Could not send message for service S123456: No Endpoint found
    for Member with id M007.
2006-05-18 12:19:42 202 Outgoing service message - destination (community) unknown
    here: C004
2006-05-18 12:19:42 295 Outgoing service message - resolved destination string (C004):
    []
2006-05-18 12:19:42 295 Outgoing service message - delivered to 0 / 0 destinations.
```

Listing 7.4: Partial G4DS logging output on node M003 for experiment stage 1 step 002c

```
2006-05-18 12:32:44 199 New incoming message
2006-05-18 12:32:45 198 -- MSG ID Z148703 | SENDER M002
2006-05-18 12:32:45 198 -- Size of msg (brutto | netto): 4862 | 529 Bytes
2006-05-18 12:32:45 198 -- Service Msg - Service test service (S123456)
2006-05-18 12:32:45 698 Access Control - message passed: M002 -> S123456 (A: g4ds.
    service.S123456)
2006-05-18 12:32:45 198 -- Service Msg - no client connected for this service.
2006-05-18 12:32:55 199 New incoming message
2006-05-18 12:32:55 198 -- MSG ID Z115529 | SENDER M001
2006-05-18 12:32:55 198 -- Size of msg (brutto | netto): 32750 | 9094 Bytes
2006-05-18 12:32:55 198 -- Control Msg - SS: Routing Engine
```

```
2006-05-18 12:32:55 698 Access Control - message passed: M003 -> C003 (A: g4ds.routing
    .route)
2006-05-18 12:32:55 298 Sending control message
2006-05-18 12:32:56 299 New outgoing message - direct delivery (M005 | C003)
2006-05-18 12:32:56 296 -- Endpoint Endpoint (E534968): MemberID is M005. Address:
    http://192.168.1.2:8080.
2006-05-18 12:32:56 296 -- Size of Data 32558 chars
```

Conclusion G4DS has proved to be able to handle faulty distribution domains on the one hand and provides reporting mechanisms for unreachable components on the other hand. In detail, the reporting is carried out the following way:

- Invalid destination strings are reported in the sender‘s G4DS log
- Unreachable services on the destination are only reported in the receiver‘s G4DS log
- Unreachable G4DS nodes are reported within the sender‘s G4DS log
- Errors are never reported to the connected application

Stage 1 - Step 03 – Access control evaluation

Step three performs checks for the G4DS system in order to test its access control capabilities.

Aim The XML-policy-based access control mechanism of G4DS is an essential component in order to protect exchanged information. I examined capabilities of G4DS to pass and block messages regarding their source or destination information.

Execution Before any of the services or applications are started, changes are applied to the access control policy files on certain nodes in order to suit one of the following use-cases:

- One node blocks application messages for any service from one other node
- One node blocks all messages sent for a certain community
- One node blocks all messages sent to a certain service from any node

Afterwards, G4DS service and chat application are started on all nodes, and attempts are made for distributing messages within the network. After shutting down application and service, corresponding logging information is moved to the corresponding folder in the output directories.

Results G4DS access control blocks traffic exactly as defined in the corresponding policy files. Access violations as well as passes are reported in G4DS logging facilities. Listing 7.5 shows partial logging data of node M001, which represents a scenario, where M001 blocks all application messages from node M003 (the corresponding access control rule is shown in listing 7.6). One can see that the application messages from M003 triggers an access violation and is not passed on to the application. Other traffic, however, is allowed. Consequently, other traffic from M003 (routed message) as well as application messages from other nodes are passed through.

Listing 7.5: Partial G4DS logging output on node M001 for experiment stage 1 step 003b

```
2006-05-18 17:19:35 199 New incoming message
2006-05-18 17:19:35 198 -- MSG ID Z598928 | SENDER M003
2006-05-18 17:19:35 198 -- Size of msg (brutto | netto): 4844 | 525 Bytes
2006-05-18 17:19:35 198 -- Service Msg - Service test service (S123456)
2006-05-18 17:19:35 601 Access Control - access violation: M003 -> S123456 (A: g4ds.
    service.S123456)
2006-05-18 17:19:50 199 New incoming message
2006-05-18 17:19:51 198 -- MSG ID Z466511 | SENDER M003
2006-05-18 17:19:51 198 -- Size of msg (brutto | netto): 31258 | 9118 Bytes
2006-05-18 17:19:51 198 -- Control Msg - SS: Routing Engine
2006-05-18 17:19:51 698 Access Control - message passed: M001 -> C001 (A: g4ds.routing
    .route)
2006-05-18 17:19:51 298 Sending control message
2006-05-18 17:19:51 299 New outgoing message - direct delivery (M002 | C001)
2006-05-18 17:19:51 296 -- Endpoint Endpoint (E145578): MemberID is M002. Address:
    193.63.148.149:2000.
2006-05-18 17:19:51 296 -- Size of Data 32752 chars
2006-05-18 17:20:12 199 New incoming message
2006-05-18 17:20:13 198 -- MSG ID Z980968 | SENDER M005
2006-05-18 17:20:13 198 -- Size of msg (brutto | netto): 6264 | 525 Bytes
2006-05-18 17:20:13 198 -- Service Msg - Service test service (S123456)
2006-05-18 17:20:13 698 Access Control - message passed: M005 -> S123456 (A: g4ds.
    service.S123456)
2006-05-18 17:20:13 698 Access Control - message passed: M005 -> S123456 (A: chat.send
    .message)
2006-05-18 17:20:14 198 -- Service Msg - passed message to connected client.
```

Listing 7.6: Additional rule for Access Control on node M001 to block application data from M003

```
<rule>
        <id>R00001</id>
        <comment>Prevent node M003 to send any messages to service S123456</comment>
        <actor type='member'>M003</actor>
        <action type='action_id'>g4ds.service.S123456</action>
        <target type='service'>S123456</target>
        <reaction type='direct'>drop</reaction>
</rule>
```

Conclusion It has been shown that G4DS is capable of carrying out access control activities exactly as defined in the access control rules within the policy files. Policies are parsed from the files at start-up-time of G4DS; all rules are applied in the order specified by their rule identifiers.

Stage 1 - Step 04 – G4DS Penetration

The last part of the stage one experiments examines capabilities of G4DS to withstand attacks which are aimed against the G4DS system itself.

Aim G4DS is intended to be used as communication platform for a variety of applications, some of them exchanging sensitive information. Therefore it is important that the G4DS system is able to protect itself against certain attack pattern, such as sniffing, denial-of-service or spoofing attacks.

Execution The way, this step was carried out depends pretty much on the attack pattern in question. Consequently, each of them is described separately:

- For the sniffing attack pattern two different kinds of examining traffic are Incorporated. Firstly, the actual traffic on the network media is captured using a network sniffer (Ethereal), which is installed on the sender‘s machine in order to simplify the process of accessing the data (in real-world scenarios, capturing would be performed on a separate machine by employing techniques such as MAC spoofing or directed routing for redirecting traffic). Secondly, small changes to the G4DS program have been applied in order to illustrate a node within the network, acting as a gateway. This is mainly for examining end-to-end confidentiality of the communication.
- The denail-of-service attacks have been carried out by sending a high load of traffic to the well-known ports of the G4DS system (2000 for TCP/IP socket communication and 8080 for SOAP communication). The payload of those messages were captured beforehand using the network sniffer Ethereal. The generating of traffic itself was carried out using the network tool netcat.
- The spoofing attacks were carried out within the two layers, the approach is using addresses. One the one hand, attempts have been made to spoof the G4DS generic member identifiers; on the other hand, IP addresses have been spoofed. In the end, both approaches were combined and the IP address as well as the G4DS identifier were modified for a message. Again, the traffic was recorded beforehand using the network sniffer Ethereal and then replayed using the network tool Netcat.

For all attack patterns in common, the G4DS service as well as the chat application were started before those were carried out. Logging information was saved after all application had been shut down.

Results As the attack patterns were very different in their nature, the results have to be examined separately as well:

- The sniffing attempts only revealed information about the nature of the message; saying it is a G4DS message (no information about the application) and the encryption algorithm utilised. Information discovered on the gateways for routed messages does not reveal any more than the destination of the message and the encryption algorithm employed (an example is shown in listing 7.7, which is the output of the G4DS routing sniffer on node M001, showing information about a message passed on to node M003).
- The Denial-of-Service attacks against the ports used by G4DS could slow down the whole G4DS system on that node very easily. Background noise messages sent only ones in a second were able to delay processing of information significantly.
- Spoofing attacks turned out to be unsuccessful due to the following two reasons:
  - Spoofing on G4DS layer caused a signature failure when processing the incoming message and processing was stopped immediately.
  - IP spoofing attacks remained unsuccessful due to impossible establishment of TCP sockets for the connections.

Listing 7.7: Additional rule for Access Control on node M001 to block application data from M003

```
*****************************************************************
05/19/06 17:38:49 G4DS Sniffer initialised


_________________________________________________________________

05/19/06 17:38:49
Destination: M003
Community: C001
raw data
--
<?xml version='1.0' encoding='UTF-8'?>
<g4ds>
  <enc>
    <algorithm>rsa</algorithm>
    <data>
      <![CDATA[789c .. loads of hex-encoded data here ... 0cab]]>
```

```
    </data>
  </enc>
</g4ds>

--
```

Conclusion After all, the resistance against attacks has been satisfying. In detail, this is again explained for each attack pattern separately:

- Sniffing attacks remained unsuccessful due to encryption employed for both, peer-to-peer as well as end-to-end communication. No actual data (payload data) could be revealed with the employed sniffing technologies.
- Denial-of-Service attacks had a big impact on the processing of messages on the attacked node. (It is believed, however, that it is impossible to protect a node or component against DOS attacks completely - in the end there could always be Distributed DOS attacks aimed against the node in question, which may not be countermeasured on the local side at all.) In the end, it was not the aim to come up with a solution for protecting a single side against DOS attacks with G4DS, but to keep the remaining parts of the network working by total avoidance of single points of failure. This has been achieved by G4DS as all remaining nodes kept working during the time-frame the node in question was attacked.
- As G4DS is employing a public-key infrastructure with asymmetric keys for authentication it was impossible to spoof the identity of a node from within the network; neither could an outsider introduce information under a known identity, nor could a G4DS insider pretend to be another node within the network.

Overall conclusions Stage 1

With the experiments carried out so far G4DS has proved to be a secure, reliable and easy to configure communication platform. Using the results from the four steps carried out for the first stage of the experiments, the following statements can be made:

- Messages are delivered throughout the network as expected and dispatched to the connected service application regarding the service identifier. Destination groups such as whole communities are broken down into its members and a message is sent to each of them. By employment of gateway technologies at the so-called Trusting Community Gateways, messages may also be exchanged across community boundaries.

- Messages with faulty distribution domains or sent to unreachable components are reported into G4DS logging facilities, which allows further (manual) handling of them.
- Access control mechanisms are in place in order to protect local knowledge. Rules for the access control mechanism are loaded at G4DS start-up time from XML-encoded policy files and applied on the fly for any incoming message.
- Certain well-known attack patterns proved to be not successful when aimed against the G4DS system. Hereby, sniffing and spoofing attacks turned out to be totally without any useful result; the denial-of-service attacks did have an impact on performance on the attacked machine, the overall system, however, could not be prevented from carrying out its work.

In the end, the installation in the laboratory environment could act as compilation for carrying out the experiments due to the number of machines and their assignment to trusting communities including deployment of gateways to exchange information between these communities. This laboratory installation will be kept and extended by more features and components for carrying out experiments for the remaining two stages.

7.3.2. Stage 2 - IOIDS

The second stage of the practical experiments evaluates, whether the IOIDS approach is capable of addressing the objectives in the context of its basic functionality. This way, message distribution and consistence into the IOIDS data repositories of the several network nodes is examined and certain tests are performed in order to measure time and size implications caused by the system.

In order to structure the second experiment stage, it was broken down into the following five steps:

1. Sending and processing of correct messages
2. Distribution of faulty messages and their error handling
3. Processing of remote events and their re-distribution
4. Different levels of event details and processing implications for certain fields within the data structures
5. Measurement of performance and overhead

This section will provide an overview, how the experiment was carried out. What the aims for each step were and results for each of them are presented including a short conclusion of them. After a first subsection, explaining the modifications applied to the laboratory environment, each of the steps will be addressed within a separate section. Afterwards, an overall conclusion for the second stage presents an overview, which objectives have been addressed in here.

Changes for laboratory environment

The basic laboratory environment with G4DS installation on all nodes have been deployed for the first stage of the experiments already. This installation will be used as a base for the second stage and extended by some features and components in order to provide facilities for evaluating IOIDS components.

The technical details for these changes are provided within the experiment protocols in the appendency B.1.2 from page 273 onwards. This is just a high-level overview of the measures applied:

- A database environment had to be deployed in order to facilitate persistence of the IOIDS application.
- As part of the database deployment, a database backend system for SOAP XML RPC has been deployed. In addition to the compulsory server software (XSM) a corresponding SOAP client has been installed as well in order to allow manual insertion of events into the event database by providing the corresponding XML description of the event.
- The IOIDS application itself had to be deployed on each node including some libraries it is depending on. Registration with the G4DS environment was carried out by applying the service descriptions and generating and exporting application keys.
- Configurations to all sub-systems were undertaken in order to make them communicating between each other.

Stage 2 - Step 01 – Correct messages

The first step of the IOIDS evaluation sends correct messages throughout the network and examines distribution, receiving and proper processing of them.

Aim It was evaluated, whether information is passed appropriately between the two components G4DS and IOIDS. Furthermore, the trigger functionality of the IOIDS system for

picking up new events from the central event database (SoapSy) automatically has been tested. Finally, it could be checked, whether event information from one node was merged properly into the data repository of another node.

Execution After bringing up all components on each node all over the network, one event has been inserted into each event data base one after the other using the SoapSy XML RPC SoapClient. Logging information from both, G4DS and IOIDS as well as database status information are relocated into the corresponding folders after shutting down all components.

Results Events have been populated throughout the network as expected; meaning, as configured in the dataengines at configuration time, every node is sending its local events off to the members of all communities, it is a member of. Listing 7.8 shows the relevant parts of the IOIDS logging information on node M001, which represents a) a new event, which is sent off to other nodes through G4DS, and b) an event, received through G4DS, to be inserted into the local data repository.

Listing 7.8: Partial IOIDS log on M001 for sending and receiving one message

```
2006-06-07 13:40:35 198 Event Trigger: Synchronise with event database.
2006-06-07 13:40:35 197 Received event (local) with id: 13 - put it into event queue.
2006-06-07 13:40:35 197 -- Event Trigger Details: 1 events received.
2006-06-07 13:40:38 298 Data engine details: check my lists
2006-06-07 13:40:41 298 -- Process local event - subsystem determination: generic
2006-06-07 13:40:50 198 Event Trigger: Synchronise with event database.
2006-06-07 13:40:51 197 Received event (local) with id: 14 - put it into event queue.
2006-06-07 13:40:51 197 -- Event Trigger Details: 1 events received.
2006-06-07 13:40:53 221 G4DS Outgoing message: Passed new message to C001
2006-06-07 13:40:53 222 -- Outgoing message details: action is ioids.write.newevent
2006-06-07 13:40:53 222 -- Outgoing message details: data size is 4423

2006-06-07 13:42:27 211 G4DS Incoming message: Received data from M002 | C001
2006-06-07 13:42:27 212 -- G4DS Incoming message details: action is ioids.write.
    newevent
2006-06-07 13:42:27 212 -- G4DS Incoming message details: data size is 4416
2006-06-07 13:42:27 212 -- Carry out 1 reactions for this remote event.
2006-06-07 13:42:34 198 Event Trigger: Synchronise with event database.
2006-06-07 13:42:34 197 Received event (local) with id: 16 - put it into event queue.
2006-06-07 13:42:34 197 Received event (local) with id: 17 - put it into event queue.
2006-06-07 13:42:34 197 -- Event Trigger Details: 2 events received.
2006-06-07 13:42:37 298 Data engine details: check my lists
2006-06-07 13:42:37 298 -- Data engine details: Processed 2 local events.
```

Conclusion It has been shown, that the collaboration between the Inter-Organisational Intrusion Detection System application and the Grid for Digital Security communication

platform handles the triggering, distribution, processing and integration at a fully satisfactory extend. No data has been lost, neither was information sent to node, which were not intended to get informed about the events in question.

Stage 2 - Step 02 – Rejected messages

With the second step of stage 2 the facilities of both IOIDS and G4DS for handling messages from untrusted parties are checked out.

Aim As much as it is important to deliver and process all desired information, it is essential to protect the data repositories from being tampered with faulty information, saying, information from untrusted parties. Access control (G4DS) and data engine (IOIDS) mechanisms should be able to identify those messages and react appropriately with dropping the message in question and report the incident.

Execution In order to carry out this part of the experiment, changes were applied to G4DS access control policies as well as to IOIDS data engine policies before running the tests, for allowing or preventing certain remote nodes to access the local data repository in write mode. After bringing up all required components on all nodes within the network, events were inserted into the SoapSy database manually using the SoapClient at different locations within the network. Distribution of those events regarding the data engine policies on the source node of the message should end up either passed or dropped on the receivers node depending on the source of the message. After inserting the events logging information from both, G4DS and IOIDS as well as database status information are relocated into the corresponding folders.

Results Both facilities the G4DS access control module as well as the IOIDS data engine processor proved to be capable of filtering messages regarding their rules they have been configured with.

Whenever the G4DS identified an incoming message to be a service message from a certain node, which has to be dropped, either because no service messages are allowed at all to be passed on from this node, or just for a service with a certain service ID in particular, it reported an access control violation to the G4DS logging facilities and dropped the message straight away.

Once, a message was passed on from the G4DS system to the IOIDS application, due to loose rules within the G4DS access control setup, the IOIDS data engine would not process and integrate it into the local data repository if the required rule is not available. (An example

is shown in listing 7.9, where the relevant parts of the IOIDS logging information on node M002 indicate that 0 reactions were carried out for a remote event from node M003.)

Listing 7.9: Partial IOIDS logging information on node M002

```
2006-06-07 16:44:53 211 G4DS Incoming message: Received data from M003 | C001
2006-06-07 16:44:53 212 -- G4DS Incoming message details: action is ioids.
    write.newevent
2006-06-07 16:44:53 212 -- G4DS Incoming message details: data size is 4423
2006-06-07 16:44:53 212 -- Carry out 0 reactions for this remote event.
```

Conclusion Knowledge protection mechanisms in place for G4DS and IOIDS make sure that information within the local data repository cannot be tampered by untrusted parties. In detail, the two measures in place are marked by the following attributes:

- The G4DS access control module is the first protection mechanism for incoming messages. Identified by the action string of an incoming message and the source member authenticated using asymmetric keys beforehand, it can filter incoming IOIDS service messages in a reliable manner using additional access control policies provided for this service.

- The IOIDS data engine is the second step an incoming message is passing through. Without an explicit rule for messages from this member, the incoming event will not be integrated into the local data repository.

Both mechanisms have reporting facilities in place (logging into files or into syslog service), which allow tracing of rejected messages.

Stage 2 - Step 03 – Remote events

It is evaluated, whether IOIDS passes on information from remote hosts to further nodes appropriately regarding its corresponding data engine policies.

Aim On top of distributing information from locally deployed event sources, IOIDS is also able to reuse information from remote sources, meaning other nodes within the G4DS network, and pass it on to further nodes. Hereby, it is very important that only information is passed on to nodes that are intended to know about those events - otherwise the trust relationship between the original source would be underminded.

Execution Certain changes have to be applied to the data engine policies on some nodes within the network in order to set them up for passing on remote event information (a sample rule for node M001 is shown within the appendix as listing B.4). Afterwards, events are generated manually for the SoapSy databases on certain nodes using the SoapClient. Finally, logging information from both, G4DS and IOIDS as well as database status information are relocated into the corresponding folders.

Results By examining IOIDS logging information, it could be proved that the nodes in question did pass on the appropriate messages exactly to the nodes defined in the IOIDS data engine policies. An example is presented in listing 7.10, which shows the processing of an incoming event from the remote source M002, the passing through the data engine and the final re-distribution to node M003.

Listing 7.10: Partial IOIDS log on M001 for passing on one remote event

```
2006-06-08 10:38:39 211 G4DS Incoming message: Received data from M002 | C001
2006-06-08 10:38:39 212 -- G4DS Incoming message details: action is ioids.write.
    newevent
2006-06-08 10:38:39 212 -- G4DS Incoming message details: data size is 4416
2006-06-08 10:38:40 212 -- Carry out 1 reactions for this remote event.
2006-06-08 10:38:44 198 Event Trigger: Synchronise with event database.
2006-06-08 10:38:44 197 Received event (local) with id: 37 - put it into event queue.
2006-06-08 10:38:44 197 -- Event Trigger Details: 1 events received.
2006-06-08 10:38:47 298 Data engine details: check my lists
2006-06-08 10:38:47 298 -- Data engine details: Processed 1 local events.
2006-06-08 10:38:49 198 Event Trigger: Synchronise with event database.
2006-06-08 10:38:49 197 Received event (local) with id: 38 - put it into event queue.
2006-06-08 10:38:49 197 -- Event Trigger Details: 1 events received.
2006-06-08 10:38:51 221 G4DS Outgoing message: Passed new message to M003
2006-06-08 10:38:51 222 -- Outgoing message details: action is ioids.write.newevent
2006-06-08 10:38:51 222 -- Outgoing message details: data size is 4416
```

Conclusion IOIDS remote event passing on is an important feature to reuse information from remote sources. In the current stage it is only the single event, which can be redistributed. After further future developments, however, event information from remote sources and local sources is supposed to be correlated and the results are supposed to be distributed, which requires strong protection mechanisms in place for keeping up with the trust relationships between members. Due to the employment of XML formatted policy files the rule engine can be extended very easily in order to address such a behaviour - the generic measures to protect the information are in place already.

Stage 2 - Step 04 – Content and levels of detail

It is examined, whether all information available for one event on the source, no matter down to how much detail, is transmitted to and processed at the receiver‘s side.

Aim The SoapSy database comprises of a number of relations which are linked to each other. Regarding the level of details available for an event IOIDS should pick up all information and insert an exact copy of the event on the receivers side no matter how many details are provided. Furthermore, it is important to avoid duplicate entries in the data repository; especially, as the same event might be received from several locations due to aforementioned redistribution features. Measures should be in place to identify those duplicates and avoid insertion of a second identical data set.

Execution The way, data is inserted into the SoapSy database using the SoapClient is the providing of a single XML document, which contains the complete set of information for a single event. Different XML documents were prepared, each of them containing a different level of detail - saying, how many related information will be linked within the database to the core event information.

Furthermore, one event was taken and only the timestamp was changed for it slightly to create another, yet very similar event.

After bringing up all the components on the nodes within the network, these created events were inserted on one node using the SoapClient. After all messages had been distributed throughout the network, nodes were shutdown and available logging information was moved to the corresponding output folder.

Results By comparing content between the data repositories of the IOIDS event source and the destination of a message it could be shown, that messages of all details are processed, transmitted and integrated completely as well as correctly. No data has been lost in any stage of this process, nor were any unknown data fields filled up with random or predefined values.

Furthermore, insertion and processing of similar or equivalent information produced the following results:

- Identical events are only stored ones in the database
- Differences within related information (such as source or destination of an attack) are only generating a new entry within their relation if information changes; new entries

are created for the relations referring to this one using foreign keys due to the change in the foreign key.

- Whenever one entity of a relation extends another entity by more information, a new entry is created. (e.g. whenever there is an entry for a computer already maintaining the IP address of this node only and then an event specifies a computer with exactly the same IP address but another host name, a new entry will be created in the corresponding relation)

The output of the SoapClient program in listing 7.11 proves this behaviour. Two events had been used, which were completely identical apart from the timestamp. The events were inserted into the database in the order a, b, a - by examining the output of the SoapClient, one can notice that the second insertion of event a comes back with exactly the same primary key (event id) as the first insertion.

Listing 7.11: Console output for SoapClient on node M004 for inserting similar messages

```
michael@grid01:~/experiment/output/M004/stage2/004/b$ python ../../soapClient.py -a
    localhost -i 01_insert_a_timestamp_alpha
<?xml version="1.0" encoding="UTF-8"?>
<RELATIONS command="INSERT_RESULTS">
        <REL name="event">
                <ATT name="event_id" primary_key="true">"49"</ATT>
        </REL>
</RELATIONS>

michael@grid01:~/experiment/output/M004/stage2/004/b$ python ../../soapClient.py -a
    localhost -i 02_insert_a_timestamp_beta
<?xml version="1.0" encoding="UTF-8"?>
<RELATIONS command="INSERT_RESULTS">
        <REL name="event">
                <ATT name="event_id" primary_key="true">"51"</ATT>
        </REL>
</RELATIONS>

michael@grid01:~/experiment/output/M004/stage2/004/b$ python ../../soapClient.py -a
    localhost -i 01_insert_a_timestamp_alpha
<?xml version="1.0" encoding="UTF-8"?>
<RELATIONS command="INSERT_RESULTS">
        <REL name="event">
                <ATT name="event_id" primary_key="true">"49"</ATT>
        </REL>
</RELATIONS>

michael@grid01:~/experiment/output/M004/stage2/004/b$
```

Conclusion From the presented results, it can be stated that information inside the IOIDS data repository is made persistent in a consistent and efficient way without allowing duplicate information to be stored. This is mainly achieved by the High-level XML RPC interface for the database, which performs those checks and only inserts information if not yet present in the database. This interface was not created by me and is not part of this project - however, loads of feedback from tests and experiments with IOIDS has been sent back into this project, helping to bring it into its current status.

Stage 2 - Step 05 – Performance and overhead

With the last step of the IOIDS stage of the experiments performance impacts and overheads produced within each layer were measured.

Aim The architecture and implementation of G4DS and IOIDS have not been developed with major focus on performance issues; it was rather a proof-of-concept implementation to justify its in practical environments. Nevertheless, it is interesting on the one hand, and makes an experiment complete on the other, to measure the impacts on time and network load generated by the components.

This step of the experiment stage is broken down into three issues, each of them addressed with actions in their own substep. The issues in question are:

- Time elapsing from an event being generated on the sender‘s side till integrated on the receiver‘s side
- Throughput for IOIDS, meaning how many events may be processed within a certain time frame
- Data overhead in each of the employed components

Execution Due to the different nature of each substep, the preparations for each of them were different as well. In detail, the following measures had to be taken before the actual tests could be performed:

- For the measurement of time elapsing rules had to be applied for the IOIDS policy on certain nodes within the network in order to make them passing on messages received from another node. This generates a chain of events being inserted into the SoapSy database, triggered and picked up by the IOIDS system and sent off to another node, which will process and insert it in turn.

- For the throughput measurements a shell script was created, which is able to insert events into the SoapSy database in a way that it always keeps a certain constant time gap between insertion of two events. This time gap will be reduced with every turn the events are inserted in order to measure the performance impacts on the nodes involved in processing the information.
- For the last part of this step a network sniffer (Ethereal) had to be put into place and activated, which is able to provide information about sizes in the different network layers.

After preparing the environment for this step, the components could be brought up on all the nodes and the required messages were inserted into the SoapSy database on one node. Afterwards, all components were shut down again and available logging and output information were relocated to the corresponding folders on each node.

Results The achieved results for each of the three different measures are:

- The time measurement results as shown in table 7.8 revealed the expected behaviour that with the increase of hops the time an event travels until it is reaching its final destination increases. The number also shows that the processing of information within IOIDS and G4DS is quite a time consuming process.
- The throughput numbers discovered that from a certain time gap on between two events, IOIDS is not capable anymore to process those events in realtime and keeps them in a queue for later processing. This delayed processing ends up in a number of more than 11 minutes when one event is sent after the other with no time gap between them at all (still there is the delay, the SoapClient is causing itself by connecting against the database).
- Table 7.9 shows the results from the measurements of packet sizes within each layer. The first entry represents a directly delivered message between two nodes. The second entry provides size information about a routed message - as a routed message encapsulates one G4DS message within another one, there are two values present for that layer.

Conclusion It has been shown that the deployed G4DS / IOIDS infrastructure is not a very quick solution for exchanging event information. Besides the justification that it has not been developed with major focus on performance, the following issues have been identified as possible reasons for this behaviour:

Table 7.8.: Elapsed time for event travelling in IOIDS / G4DS

Source	Dest.	Hops	Sent	Received	Time elapsed
M004	M005	1	08:21:18	08:21:40	22s
M002	M003	2	08:23:10	08:23:39	29s
M002	M005	3	08:25:17	08:26:01	44s

Table 7.9.: Packet size within the different network layers

	IOIDS	G4DS	TCP
Direct msg	4421 B	3053 B	13856 B
Routed msg	4421 B	3021 B	
		17252 B	58010 B

- G4DS as well as IOIDS have been programmed using the programming language Python, which is an interpretive language; hence not very fast in execution by nature
- Messages are wrapped and re-wrapped in several layers; ones within the IOIDS application and several times inside the G4DS layer. This is a time consuming process; especially because each wrapping is incorporating XML technologies such as DOM, encoding technologies for the payload (hexencoding) and compression technologies for reducing the amount of data to be sent. (see section 5.3.2 for details)
- The trigger functionality within IOIDS for picking up new events has been implemented using a software trigger, which retrieves new events on frequent lookups in the database. (see section 6.2.3 for details).
- The machines utilised for the experiments are not of high specifications. With the installation instructions from the first stage (section 7.3.1) an overview had been given - the technical details are presented in tables B.7 and B.8 within the appendix.

The mentioned issues prevent IOIDS / G4DS to distribute knowledge in (near) real time if under high load. However, as data is kept in queues, it does not get lost and will be processed and distributed at a later time, when the amount of new messages has decreased.

The overhead, each of the layers is producing is reasonable and due to compression measures in place, the payload for a G4DS messages is smaller than the one of an IOIDS message, which is carried within the G4DS message. Due to encoding measures, G4DS is employing, and the

additional overhead the low-level protocol such as SOAP are producing, the payload in the TCP segment is about four times the size of the G4DS payload.

Overall conclusions Stage 2

With the second stage of the practical experiments the mode of operation of the Inter-Organisational Intrusion Detection System application in correlation with the already evaluated G4DS communication platform and its features as well as limitations including performance measurements were examined. Following the results from these experiments and correlating the statements from the low-level conclusions allow us to make the following statements:

- The whole process, an event is travelling from being picked up from the SoapSy database, passed on to the IOIDS data engine, which in turn may send it off to the G4DS system if population is defined within the IOIDS policies, passed through the G4DS system including its access control mechanisms and processed on the receivers side of the message by passing through the same components in opposite order in order to end up in exactly the same set of information on the receiver‘s SoapSy database, was shown to be working.

- There is an easy and straight forward way to configure the reactions of the IOIDS data engine using XML encoded policy files. The processing of events from local sources may be controlled by those as well as the one of events from remote resources. Event information from remote sources may be reused and passed on to another distribution domain without tampering trust relationships between nodes.

- Information within the distributed IOIDS system is kept consistent. An event at the receiver‘s side is an exact copy of the event available in the database of the sender. Event persistence is performed in a reliable and efficient manner.

- The correlation of IOIDS and G4DS turned out to be not a very performant way of exchanging event information. High load of events to be processed and distributed prevents the system to carry out its work in real-time, not even near-real time. However, the following statements can be made to put these results into perspective:

  – IOIDS is by its nature not supposed to exchange low-level event data and this way establishes a database on each node, which is an exact copy of the other‘s data repositories. Instead, intelligent rules for the data engine shall enable the

IOIDS system to correlate information and only distribute results or information of a more abstract nature between the nodes.

- Due to the high grade of configuration options for both IOIDS and G4DS it is also possible to layout machines in kind of a hierarchical manner, meaning to have nodes in the bottom layer exchanging quite low-level and detailed raw event information and the more a node is placed up in the hierarchy the less details are dealt with.
- All components for one side had always been placed on the same machine (G4DS, G4DS data base, IOIDS, SoapSy database), which will not mirror a real-world scenario in the end either.

Nevertheless, the IOIDS installation within the laboratory showed that intrusion detection event information can be shared within a network, when the nodes are grouped in so-called communities and base their distribution as well as processing decisions using rules on these and the trust-relationships established within them.

The laboratory environment in place now will be reused for the last stage of the experiment, explained in the upcoming section. Certain additions have to be applied in order to examine the fully deployed IOIDS infrastructure, using real-world data generated by third-party intrusion detection system sensors.

7.3.3. Stage 3 - IOIDS data integration - Preparation

In the last stage of the practical evaluation process the IOIDS architecture is to be examined as a whole and, this way, its features and abilities shall be compared to similar or related projects, which were deployed in the laboratory environment as well.

The following sections give details about the changes in the laboratory made for the third stage, whereby the changes for the IOIDS infrastructure are explained first. Afterwards, the related projects with their deployment procedures are introduced. The environment deployment is finished off with an overview of all working components.

For all approaches in common, the following assumptions are made, which should be hit by their configurations:

- A high-level installation shall be deployed, meaning that one single node represents a whole organisation; communication between to nodes is always considered to act as traffic between two organisations we want to simulate. Consequently, all components required for a single-side installation have to be deployed on the same computer.

- Nodes within the network have different roles assigned, that means in particular that nodes M002, M004 and M005 act as data sources.
- Every node within the network maintains its own data repository and acts as a data sink.
- Domains or so called communities are deployed as shown for the previous two stages. Consequently, we are still dealing with the three communities C001, C002 and C003.
- Nodes M001 and M003 have the responsibility to pass on information between domains; this behaviour might be called depending on the approach gateway functionality or relaying.
- Node M006 will be used as a penetration node only for this stage; meaning no IDS software components need to be deployed on it.

Deployment of IOIDS environment

As most of the components have been set up in the previous stages of the experiments, only the following few actions had to be undertaken to configure IOIDS for this stage:

- A snort distribution had to be installed on the source nodes M002, M004 and M005, which had to include the PostgreSQL output plug-in. The database management system had to be prepared for holding snort event data and snort had to be configured to log into the locally available database.
- The so-called SnortDB-To-SoapSy converter had to be installed on the same machines in order to pick up snort events from the snort database and migrate them into the central SoapSy database. (some more information is given in the following paragraph)

On top of these changes, a network penetration tool had to be installed on the node in the managed network 192.168.1.0/24 for triggering alerts within snort when attacking the capturing nodes. The nessus daemon was installed on M006; the GUI based nessus client on M004.

SnortDB to SoapSy Converter There was a need to insert real event data into the SoapSy database to be picked up by IOIDS and passed around within the network. Since the SoapSy project itself has still been in its early stages, there was not yet any tool available doing this job. Consequently, a light weight application was developed by me as part of the project, to generate real-world event data into the SoapSy database. The application named SnortDB-To-SoapSy converter is marked by the following attributes:

- It is an application written in Python, which picks up event information from a SnortDB (PostgreSQL) database using software-trigger-functionality and after re-formating the data, puts it into the appropriate format into the SoapSy database using the XML RPC interface named XSM.

- Whenever possible, all data available in the SnortDB-database will be mapped into fields of the core within the SoapSy database.

- All remaining data goes into the SoapSy extensible for Snort - consequently, not a single piece of data can get lost when migrating from the Snort database into the SoapSy database.

When developing this program, I was aware of the fact, that configuring snort to log into the database, picking them up from there using a software trigger and passing them on into the SoapSy database using XSM is not the most performant solution for creating real world data in the repository. However, development of these converters is beyond the scope of the project and due to the structure of data, provided by the snort database, it was a very straight-forward way to migrate into SoapSy. (Faster technologies could have been text analyser for snort event output files or even a snort output plug-in, which would directly log into the SoapSy database.)

Deployment of Prelude environment

A normal prelude deployment comprises of sensors, managers (which may interact in form of relay managers), a database backend and a frontend called PreWikka. A sample deployment scenario taken from the Prelude handbook (ThePreludeTeam (2006)) is shown in figure 7.2.

The installation of prelude components has been carried out using the Prelude handbook (ThePreludeTeam (2006)) available on the internet. With the idea in mind to demonstrate a deployment of large-scale environments, major focus has been drawn on the installation of managers and the sensors are acting more like a tool for generating some real world event data.

In detail, the following components have been installed the following way (the technical details are shown in section B.1.3 within the appendixes):

- Besides node *M006* all nodes within the laboratory environment (*M001* - *M005*) were equipped with a Prelude-Manager installation. In order to carry out this, the prelude libraries had to be installed first. (M006 has been left behind here as it is acting as a penetrator for the stage three scenarios)

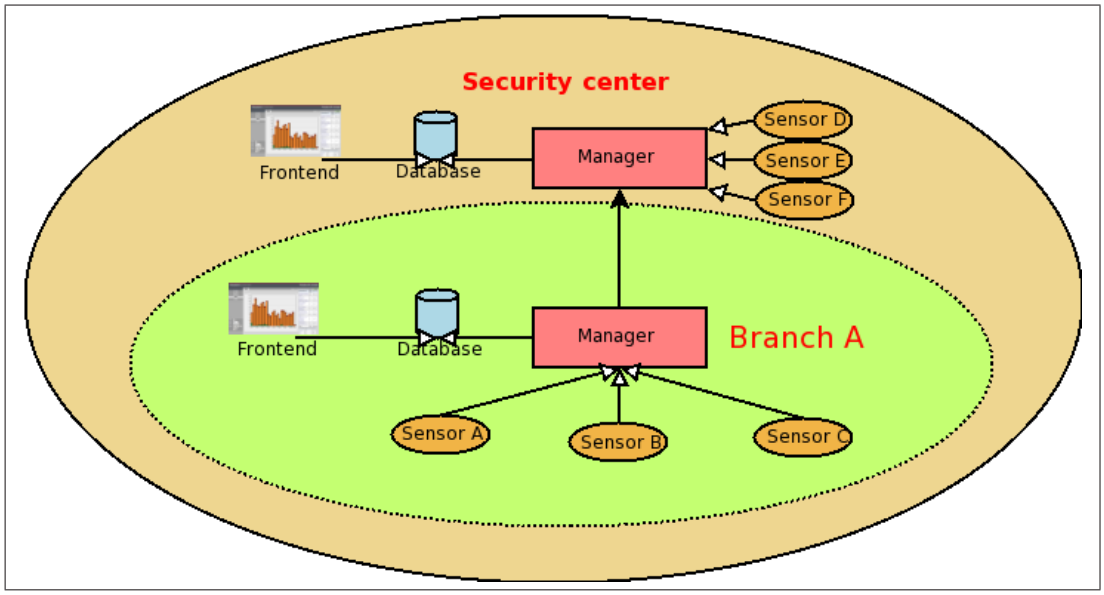


Figure 7.2.: Prelude deployment scenario (src: ThePreludeTeam (2006))

- A database (PostGreSQL) had to be installed for each of them. The database management system for each of them was installed already due to databases installed for former experiment steps; consequently, the database is always running on the same machine. The prelude database libraries were installed, so that the prelude manager was able to connect to the prelude database backend.

- The nodes with *G4DS* ids M002, M004 and M005 were equipped with an additional snort installation, which is able to output into the prelude system by using a prelude output plug-in.

- The prelude output plugins of snort were registered with the corresponding local prelude manager and keys and certificates were exchanged.

- On node with id M001 the web-based prelude analysis console *PreWikka* was installed. The therefore required database was installed on the local PostGreSQL database management system.

- Certain managers were prepared to be able to act as relay managers between each other; meaning, certificates were exchanged and rules were prepared. These are explained in more detail in the execution of the stage 3 experiments.

Deployment of SnortNet environment

For SnotNet the installation process could not be carried out as straight forward as expected. While the deployment scenario for the snort components could be planned well in a similar approach as explained for the Prelude environment using the SnortNet Manual (Fyodor (2000)) (an example deployment taken from this manual is shown in figure 7.3) the implementation and deployment of those turned out to be rather difficult.

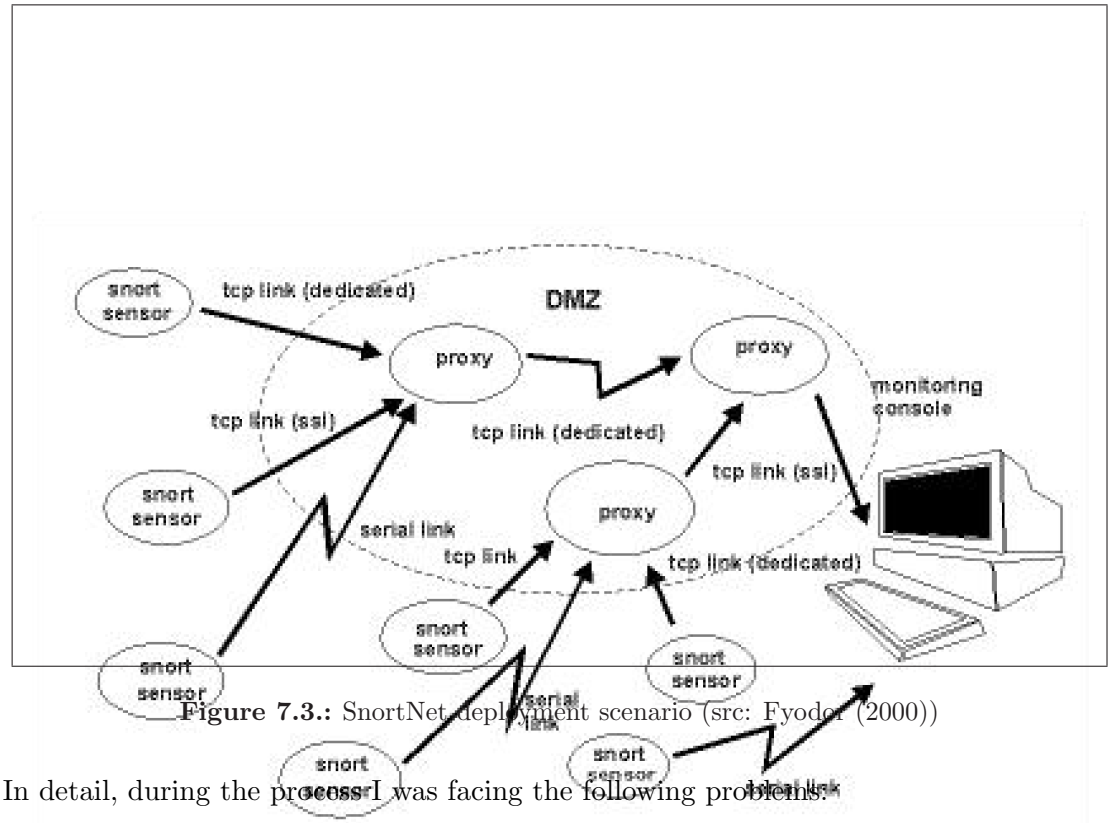


Figure 7.3.: SnortNet deployment scenario (src: Fyodor (2000))

In detail, during the process I was facing the following problems:

- Besides the initial SnortNet publication (Fyodor (2000)) from 2000 there is no documentation available; in particular does this include deployment instructions.
- The source code for the project is not made available in a proper way and had to be downloaded from the Snort CVS repository in older branches.
- Nevertheless, the source code had not been touched for more than 5 years - meaning no updates or maintenance works were applied.
- Consequently, the SnortNet reference implementation cannot be applied to current versions of the snort intrusion detection system implementation. However, due to missing

documentation there could not be found a way to apply it to an older version of snort either and use that one instead.

- The project is discontinued in a way that no further information is available. The project website (*http : //snortnet.scorpions.net/*) is not available anymore without giving any information about referrals.

Facing all these problems, where there seemed no way to overcome them in order to carry out experiments with the implementation, it was decided to discontinue experimenting with SnortNet and looking out for available and adequate alternatives instead.

From the list of available approaches in the context of distributed intrusion detection systems as discussed in very detail in 2.4 and compared in the experiment definition in section 4.3.1, the *AirCert* project was chosen for laboratory deployment despite its high complexity. Consequently, the following section describes the efforts and problems on deploying the implementation of this approach in the laboratory environment.

Deployment of AirCert environment

With the idea in mind to deploy a scenario very similar to the ones of the IOIDS and Prelude infrastructure, available documentation has been researched in order to identify required components and their responsibility within the overall AirCert infrastructure. An AirCERT handbook is available for these purposes, which describes, how components interact on the one hand and gives deployment guidance on the other hand.

In figure 7.4 is shown a peer-based deployment scenario taken from this handbook. However, AirCERT is not limited to peer based deployments and can support hierarchical arrangements of components as well.

Besides the handbook, very little information was available for deploying the components and make them interacting. In detail, I was facing the following problems during my attempts of deploying an AirCERT infrastructure:

- Although the AirCert project team continues their work and is still progressing on implementation, it seems to focus on another (somehow related) project called *NetSA Aggregated Flow*[1] right now, so that the latest sources available for AirCert components have aged two years already.

- Released software is either in alpha or beta status and proved to be neither very reliable nor easy to configure. Again, documentation describing the way, how to use the

[1] *http : //aircert.sourceforge.net/naf/index.html*

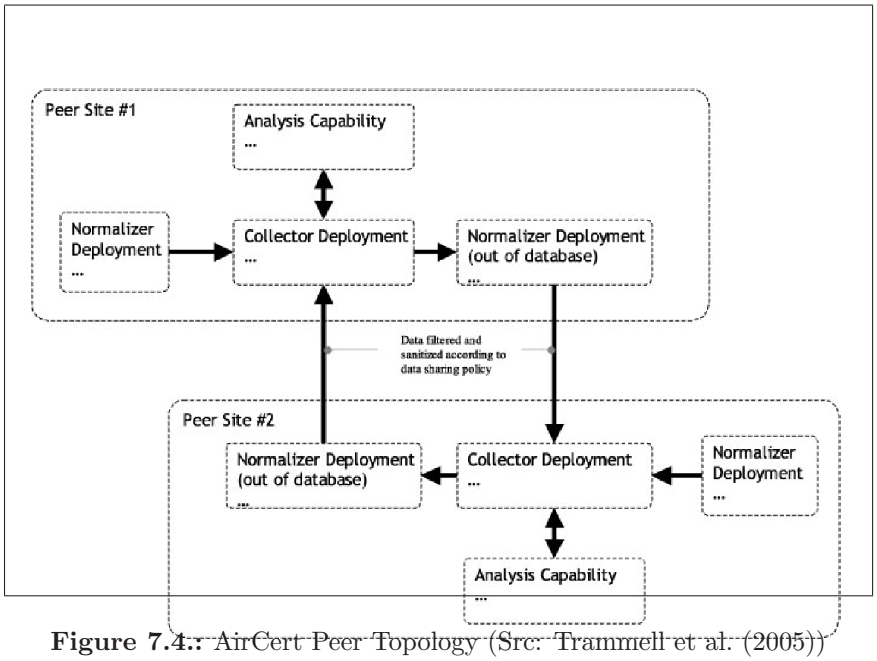


Figure 7.4.: AirCert Peer Topology (Src: Trammell et al. (2005))

components (especially the way command line arguments have to be passed to them) is incomplete, unsatisfying and has to be pulled together from the different versions. Examples were not provided at all.

- There is no contact information available for the overall project. Requests to authors of certain components remained unreplied.

- The entire infrastructure is very complex - there is no opportunity of carrying out a light-weight deployment of AirCert. In detail, the components for a single side within the infrastructure comprise of (talking about the simplified version, which does not include the components for database input processing):

  – A snort installation, configured to log text-based output into the local filesystem

  – The AirCERT *rex* Text File Normaliser for processing the Snort output and creating XML formated output from it (SNML)

  – The AirCERT *twrap* module for archiving and packing information before transmission

  – The AirCERT *dredge* DAV client, which is in charge of transmitting the collected data to the data collector

  – An Apache web-server installation, in charge of receiving this information using

modules for WebDAV (mod_dav), certificate authentication (mod_ssl) and URL rewriting (mod_rewrite) and storing it in a local directory.

- The AirCERT *untwrap* modules for unpacking and extracting the information from the archives.
- The AirCERT *pathogen* Relational Database De-normaliser, which processes the received event information in SNML format and puts it into the database.
- (optionally) the *ACID* analysis console for AirCERT, a web-based frontend for displaying the content of the event database.
- An *openssl* infrastructure, where each of the nodes acts as a certificate authority. Certificates have to be derived for each sensor and for each collector. They have to be converted into certain formats (x509 for CAs and pkcs12 for clients) as AirCERT components are not prepared to handle the standard formats created by the openssl implementation.

When installing the aforementioned components, a major issue besides incomplete documentation was the lack of up-to-date SQL scripts, preparing the databases for storing required information. Manually created SQL scripts for updating the database relations created from error messages during execution could not totally eliminate these problems.

- Although, the AirCERT handbook provides quite a good entry point of documentation, there is big lacks for gaining information about the components. Often, readme files and manuals only exist in form of unmodified templates; code examination becomes very difficult due to lack of comments, giving information about what certain modules or functions are responsible for.

The mentioned reasons were forcing me to give up on experiments with AirCert after about a week of unsuccessful deployment attempts and focus on the remaining projects instead. The presented list of representative projects in the context of distributed intrusion detection (see section 4.3.1) does not offer any other item, which is suitable for practical evaluation for the IOIDS project, mostly due to missing implementations.

Consequently, the comparison part of the practical evaluation process for IOIDS will be based on one further product only, namely Prelude IDS. This does still provide meaningful information, since Prelude is an up-grown and popular approach in this context. Finally, a further comparison with other approaches has been done by feature comparison in the beginning of this chapter (section 7.2) already.

After all, the devices in the laboratory environment are running the services and maintaining databases as described in figure 7.5.

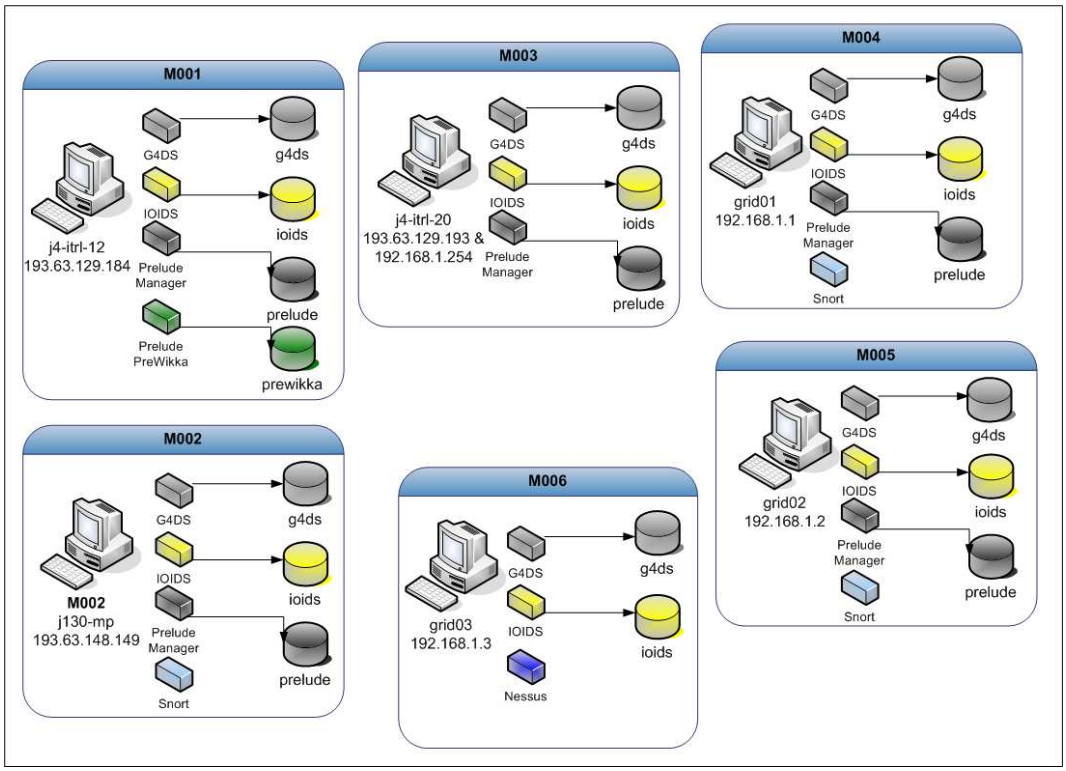


Figure 7.5.: Overview of components and databases on laboratory machines

7.3.4. Stage 3 - IOIDS data integration - Execution

The following five subsections explain how the last stage of the experiments have been carried out and what conclusions could be derived from the given results. As maintained for the former stages, this one has been broken down into steps as well; which are:

1. Distribution of correct event information
2. Generation of high load of event information on certain nodes within a managed network
3. Availability issues and denial-of-service resistance
4. Data engine and access control
5. Performance and benchmarking

Each of those steps is discussed in its own subsection over the upcoming pages.

Stage 3 - Step 01 – Distribution of correct event information

The third stage of the experiment is started up with processing of event data for a machine placed on the internet.

Aim It shall be examined, how the approaches behave when requested to process event information, which is common for a network IDS placed on the internet. This shows their behaviour under ordinary load and examines, whether the whole process from event triggering through distribution and processing up to integration at remote sides is carried out appropriately.

Execution In order to carry out this step, the network sensor had to be placed on a node outside the managed network; meaning directly connected to the internet. This was achieved by using node M002, which is on the university's network and has a public IP address. Certain distribution rules had to be set up (in the IOIDS data engine policies as well as the Prelude-Manager configuration files) in order to enable sharing of event information between the nodes. Three runs over five minutes were executed with each IDS, whereby these ones were alternated in order to achieve conditions as similar as possible for the two of them.

Results An average number of 40 events was generated by the snort sensor for each of the 6 runs. It could be shown that both approaches handles the incoming events properly; meaning, an entry was inserted into the local data repository and distribution was performed to node M001, as set up for this stage. Figure 7.6 shows the results presented on the receiver side of the Prelude relaying process M001. (A screenshot with the details of the given events is shown in figure B.4 within the appendixes.)

Conclusion Both approaches IOIDS as well as Prewikka showed to be capable of processing all the events captured by snort. They were made persistent in the local event database and passed on to the remote destination M001, which contained exactly the same event information as M002 after execution of this stage.

Stage 3 - Step 02 – High load

The second step of this stage examines the approaches in the abilities to maintain working under high load of incoming event information.

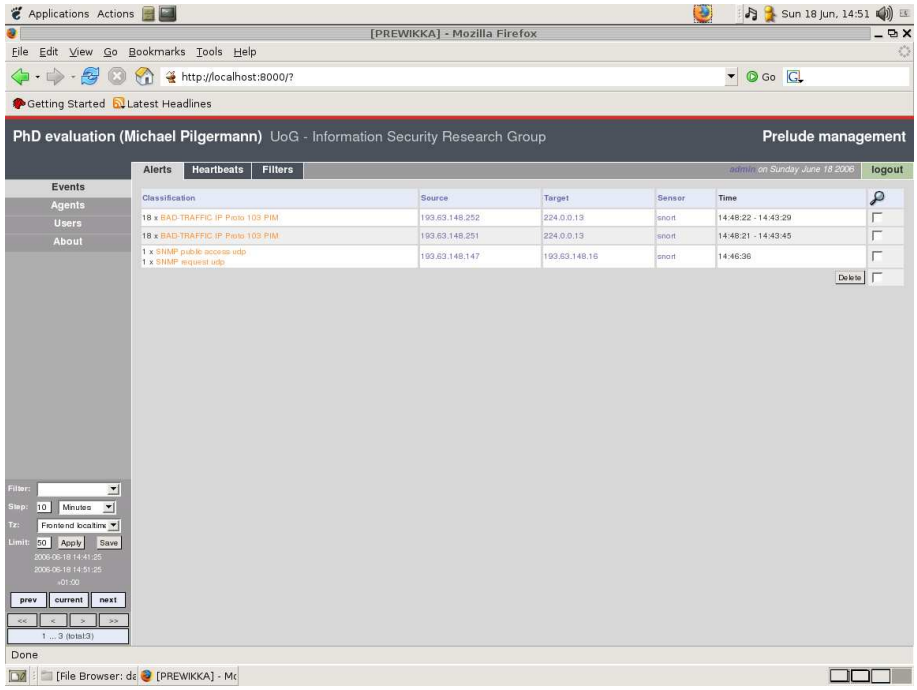


Figure 7.6.: Prelude analysis console *PreWikka* event output

Aim It is an important issue that data does not get lost when the approach in question is requested to process a high number of events. Furthermore, it was an objective of this project to deliver event information in real-time or at least near real-time. It shall be examined, from which load on this issue cannot be addressed anymore by the approaches in question.

Execution The network sensor snort is now placed inside the managed network and the alerts within snort are triggered by running a network analysis tool (Nessus) against the capturing nodes. The nodes within the managed network are setup to report events between each other (as far as possible). In order to generate a high number of events on the snort sensors, three full nessus scans (including all available plugins) were targeted against the capturing nodes (the configuration, execution and displaying of results of nessus is shown in figures B.1, B.2 and B.3). Each nessus run was expected to last about 5 minutes and in total the three should generate more than 1000 events on each capturing device.

The test had to be cancelled for the IOIDS approach as it was not capable to process this vast amount of event information. In detail, the following components were taken down after certain conditions:

- The SnortDB-To-SoapSy converter was shutdown on both nodes (M004, M005) after

it had inserted 750 events in average on each of them. This was several minutes after the last event had been triggered and inserted into the database by the snort sensor.

- The IOIDS system together with the G4DS system were shut down after about one hour of processing the events as, even after 45 minutes passed since the last event had been inserted into SoapSy, IOIDS was not able to catch up with the amount of event data.

Results Prelude IDS was capable of dealing with the high amount of event data without showing any performance impact (figure 7.7 presents the results of the events, which were received on M003 from M004 and M005). IOIDS, in contrast, was in serious problems when dealing with this amount of data - it was falling behind with processing of the incoming data in a way, that even after waiting about 45 more minutes after arrival of last event, not all events had been processed yet.

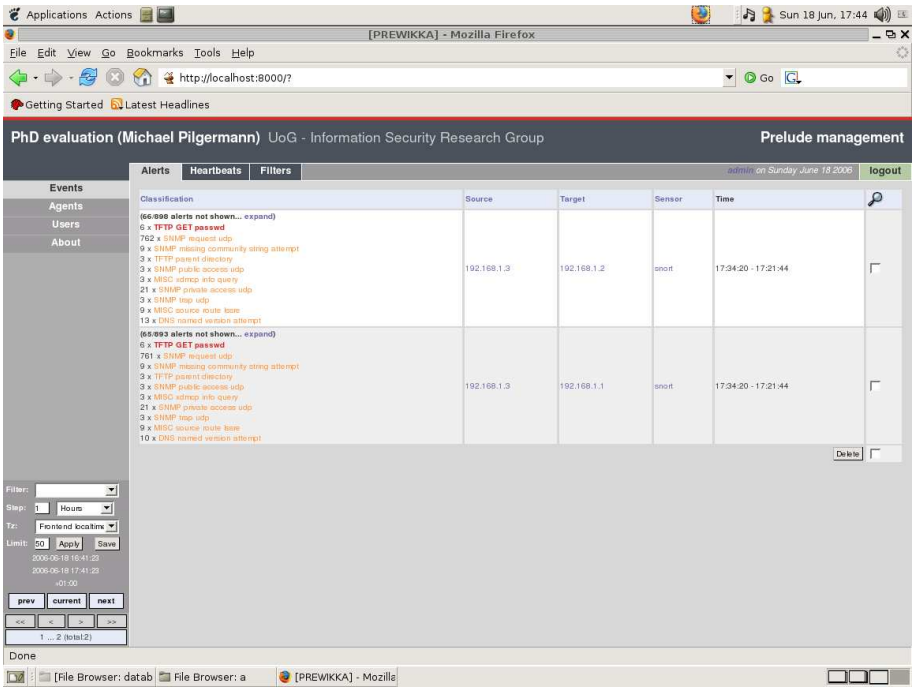


Figure 7.7.: Prelude analysis console *PreWikka* event output

Conclusion The issue of not missing out on any data could be satisfied by both approaches properly. The time passing between the arrival of an event and its processing and distribution is a big difference: when Prelude seemed to process and distribute all information

in near-realtime, IOIDS could not even post-process the event information when not giving it anything else to process for another 45 minutes. One minor issue was identified in the distribution rules for the events: when Prelude was only distributing from M004 to M003 and M005 to M003, IOIDS was always distributing peer-based in all directions. Further discussion of this issue and problem identification are made in step 4 of this stage, when dealing with availability issues and denial-of-service resistance.

Stage 3 - Step 03 – Denial of Service and availability

Step three looks to address availability issues of the approaches by switching off certain components within the network and examine the capabilities of the approaches to keep the overall system working.

Aim It is impossible to deploy a system, where each component is completely resistant against Denail-Of-Service attacks itself. The aim of this step is not to check the DOS resistance of single components (as it is not believed to be the major issue), but on the availability of the overall system, whenever certain components are switched of.

Execution The approaches comprise of different components, namely the IDS component and a communication component. Taking the network as one further target, there are four use cases to cover for denail-of-service attacks:

- All components are fully functional.
- The IDS component is taken down.
- The communication component is taken down.
- The network interface itself is penetrated.

As there are nodes within the network acting as so-called gateways, penetration of them should have another impact when targeted against them. Consequently, each of the aforementioned use cases is carried out twice; firstly for a data source node and secondly, for a node responsible for passing on information between communities.

Ones, the component in question has been taken down, a port scan will be targeted against a capturing node in order to generate a reasonable amount of event information. It was examined, how the remaining components could keep up with their work and, furthermore, what recovery procedures were in place for the approach after the penetrated component has been brought up again.

Results Prelude IDS benefits a lot from its built-in recovery mechanisms. Whenever any component within the path the information is supposed to travel is not reachable, it buffers the events and tries again and again to deliver them. This is the case for the prelude manager on the source node (where the event information is buffered within the Snort prelude output plug-in) as well as the prelude manager on the gateway, which got the information passed on from the source prelude manager, running the snort sensor.

The IOIDS implementation benefits from its database software trigger implementation in a way, that messages from the SoapSy database are picked up immediately after bringing back up the IOIDS program. As there is not yet any real recovery mechanism built into IOIDS or G4DS, messages, which could not be delivered due to G4DS problems get lost.

Conclusion Prelude‘s manager architecture comprises of a single component, the prelude manager only. Consequently, a problem with this single component prevents the whole system from working on the local node. If the node in question is not on the gateway path of event information, remaining nodes are not impacted by problems on one node. Furthermore, sophisticated and reliable fall-over mechanisms enable Prelude to reestablish a consistent state on the node after bringing it back up.

The IOIDS approach has not yet implemented any real recovery mechanisms. However, due to its database implementation it is able to post-progress event information if the IOIDS system had gone done by any reason for some time. Furthermore, the modularity of the system with separation of IDS component (IOIDS) and communication component (G4DS) can minimise the impact in case of problems. Theoretically, there would not be a big problem to implement recovery mechanisms within IOIDS, which holds events, which cannot be delivered when G4DS is down, and sends them off via G4DS after problems have been resolved.

In the end, the two approaches follow a different attitude as well. As Prelude is considered to be implemented for inside deployment, IOIDS is supposed to be deployed on a wider area. Prelude can only deliver messages in push mechanism; IOIDS however could after reestablishment of connections send a query to certain nodes in order to get events from a certain timestamp onwards.

Stage 3 - Step 04 – Date engine and access control

It is examined, whether the approaches are able to adopt easily rules for distributing to and integrating knowledge from certain remote nodes.

Aim The whole idea of exchanging event information between organisations is based on trust and trust relationships between the involved parties. It is therefore essential, that nodes can

configure the behaviour of their deployed system in a way that information is passed on to a certain set of destinations on the one hand, and that knowledge from certain remote sources shall be processed in a certain way on the other.

Execution For the execution of the test the components have to be reconfigured again and again in order to suit the different use cases, that shall be mirrored by them. When we consider two different distribution domains α and β, these use cases are as follows:

- Accept all traffic from domain β; drop events from domain α
- All locally generated events from event source A shall be distributed to β, local events from source B to domain α instead
- All event information received from β shall be redistributed into domain α
- On top of the previous rule, the other direction shall be addressed as well and event information shall also be passed on received from domain α into domain β
- Protection mechanisms shall be tested by only passing on information from domain β to domain α if the knowledge chunk in question is not classified as a private piece of information
- All information from domain α shall be dropped straight away

Certain sample events had to be created in order to insert them into the system manually. Ones, all configurations had been applied for the use case in question, the experiment was carried out for the two approaches one after each other. Log files and other results were relocated into the corresponding output folder.

Results As far as possible to configure, the two approaches achieved the desired results. Prelude IDS, however, could not be configured at all to carry out the following two items out of the use cases list:

- The second use case with the distribution decision based on the type of the event cannot be mapped into Prelude as is does not distinguish between different types.
- Prelude IDS does not support classification or any other kind of protection levels to be assigned to events. Consequently, it was not possible to configure it in a way to base distribution decisions on those values.

Another problem for Prelude IDS was the distribution of event information in two directions. This resulted in bouncing of events between the two nodes in question and the prelude-managers had to be switched of in order to stop this behaviour.

Conclusion The differences in nature between the two approaches have become obvious in this step. As IOIDS was developed with bi-directional, inter-organisational event exchange in mind, Prelude IDS rather seems to be supposed to be deployed inside organisations and better supports hierarchical layout of managers. As the use-cases chosen for this step have to map the project's aims of inter-organisational information exchange, Prelude shows its down-side for these purposes.

Besides the problems discussed in the results sections already, Prelude does not support easy re-configuration of distribution and integration rules. When the distribution domain may be changed still quite easily by reconfiguring relay managers within the configuration files, the integration of event information has to be configured by messing around with the key list of trusted sources. Whenever a source shall be disabled, it cannot just be marked as such a one; the key has to be removed from the corresponding list using Prelude admin tools.

IOIDS in contrast is able to satisfy all the requirements coming with the use cases completely. Not only did the results show exactly the desired figures, but the configurations could also be carried out very easily by just changing IOIDS or G4DS policy files.

Stage 3 - Step 05 – Benchmarking

In the very last step of the experiments it was examined, how the different approaches perform in the view of time and size overhead benchmarks.

Aim Although there was not payed much attention to performance issues when developing the IOIDS approach, the tests in here shall give a brief idea of time and size impacts caused by it in comparison to the Prelude IDS. Figures were gathered and analysed for the following issues:

- Maximum number of messages being processed within a certain time frame
- Time elapsing for the full processing, distribution and integration of a single event
- Size overhead for the different network / application layers

Execution For this step it was essential to be able to generate an exact number of events within a given time frame. A simple snort rule was created for generating a single alert,

which could be triggered with a simple netcat command (see section B.1.3 on page 293 for details). By putting this given netcat command inside a loop and create a time gap between two netcat executions it was possible, to generate exactly a pre-defined number of events within snort for a certain time frame.

Further adjustments had to be done on some of the nodes for their IOIDS data engine or Prelude Manager policies in order to enable passing on of messages into further domains. A timestamp was always taken directly before each run in order to measure elapsed time.

Size overhead within the network layers as well as confidentiality issues have been addressed by placing a network sniffer (Ethereal) on certain locations within the network and capturing the corresponding traffic for the approaches in question. The corresponding Ethereal output dumps were relocated together with other logging output into the corresponding folder within the output directory.

Results As the nature of the different parts of this step are rather different, the results have to be examined separately, too:

- The measurements of time elapsing for an event being processed could be taken from timestamp information within log files for IOIDS. Prelude does not include the information for the different hops within its logfiles; so I could only use timestamp information from last accesses to the logfiles for this project. The results for this step are presented in table 7.10.

- The possible throughput was measured in two ways; firstly, how many items got lost, and secondly, how much time did the approach in question post-progress event information since no new events had arrived for a certain amount of events within 2 minutes. The results for this part are presented in table 7.11. There are no values for IOIDS from the time gap of one second on because it was shown in the tests before that IOIDS was not able to process the number of events anymore. The test with 240 events per 2 minutes was executed twice due to the surprising drop of events in the Prelude system. However, the second run produced exactly the same number of lost events.

- The test for size overhead and confidentiality issues were carried out in one go. As both approaches are employing encryption technologies for all their communication, data could not be extracted from the sniffing output. The numbers for the size of event information sent over the network are shown in table 7.12.

Table 7.10.: Time elapsing for event distribution

	Sent	Local	1st hop	2nd hop	3rd hop
IOIDS	11:54:01.505	3.5s	24s	40s	59s
Prelude	12:02:09.797	$<$ 1s	$<$ 1s	$<$ 1s	$<$ 1s

Table 7.11.: Maximum number of processable events

			IOIDS		Prelude	
Run number	**Time gap**	**Events / 2 mins**	**Post-process**	**Lost**	**Post-process**	**Lost**
1	10s	12	1 min	0	0s	0
2	5s	24	3 min	0	0s	0
3	2s	60	12.5 min	0	0s	0
4	1s	120	-	-	0s	0
5	0.5s	240	-	-	0s	126
6	0.5s	240	-	-	0s	126

Table 7.12.: Packet size implications of approaches

	IOIDS	Prelude IDS
Single Event	15.900 Bytes	1.900 Bytes

Conclusion The evaluation of the results for this step brings to light a big difference between the two approaches. The only thing the two of them have in common is the confidentiality for the exchanged information as capturing and analysing network traffic from neither of the approaches can reveal any useful information.

In the view of distribution speed, Prelude IDS can prove significant superiority due to its direct connection between several managers and the slim architecture with a single component on each side only. The only down-side can be seen in the dropping of messages ones it is put under very high load. IOIDS comprises of a totally different architecture and has to insert each event into its database first of all, and pick it up using a trigger afterwards before it can be passed on to the next node. Several components are involved in this process and employment of wrapping, compressing and encryption technologies in several of these layers slow down the overall process significantly. It can be concluded that the two approaches are prepared to handle different deployment scenarios: Prelude IDS is performing very good results by passing around event information inside organisations due to its persistent communication link between managers and the promptness, events are passed on to the next manager. IOIDS, in contrast, reestablishes a communication channel any time information shall be passed on. Much more attention is drawn to trust issues than the contemporary delivery of knowledge.

Overall conclusions Stage 3

Merging together the results from the last stage of the experiment and mirroring them against the objectives, as laid down in chapter 3, the following statements can be made concerning the overall performance of IOIDS with all its components:

- The central issue of delivering IDS event data within communities and under certain aspects across community domains could be satisfied completely by IOIDS.

- Strong authentication and encryption mechanisms together with a policy based access control approach within IOIDS guarantee protection of knowledge and sharing of knowledge with trusted parties only.

- The XML formatted IOIDS data engine policy files allow a very easy and straightforward way of configuring the IOIDS behaviour.

- The major issue of availability in the sense of total avoidance of single points of failure could be satisfied as it was shown that taking down single components does not impact the overall performance of IOIDS.

- IOIDS has turned out to be a very structured approach for classifying information on the one hand and making decisions about distribution on the other. These two responsibilities can be broken down into their parts, which allows direct employment of available policies.

- Correlation of information so far is performed by integrating event information from many locations into a single database. The actual process of correlation is left to further analysis tools, which would insert novel information chunks, gained from this correlation, into the database, in order to get it redistributed by IOIDS.

- The reporting bit for IOIDS has been excluded from the implementation itself since there is loads of research ongoing, dealing with visualisation of information, available inside the SoapSy database.

- IOIDS does not directly integrate with other IT security relevant approaches out there to date. However, the research for it has been carried out in close contact with related projects within the research group, which are all together looking to bring up a solution for intrusion detection systems, which is able to gather, store, distribute, integrate, analyse and visualise intrusion detection system event data in an intelligent and efficient manner.

- The distribution of knowledge in realtime cannot be satisfied by IOIDS. We cannot even call its behaviour near realtime; especially when put under high load. Other approaches such as Prelude IDS proved that quick distribution is possible in general; consequently, for a proper (and not proof-of-concept) implementation certain parts of the architecture would need to be redesigned.

It is believed that with the experiments carried out in the laboratory environment a great set of real-world scenarios could be simulated. Besides the time constraints, which were set for the project, the IOIDS implementation seems to satisfy the needs laid down in the objectives earlier on.

More information has been provided in the feature comparison when evaluating IOIDS against a number of similar or related approaches in the DIDS context. Overall conclusions, that can be drawn from all the results given within this chapter are drawn in the following section.

7.4. Evaluation of results and overall analysis

The results gained from the experiments and the evaluation of available documentation has shown that the IOIDS approach is able to satisfy the objectives laid down beforehand and, this way, proves the hypothesis. In detail, the outcome of the analysis can be described as follows:

- By using G4DS the approach communicates in a very secure and reliable manner with strong authentication and integrity measures in place. Total avoidance of single-points-of-failure provides a very high availability of the overall platform.
- A policy based access control mechanism is able to satisfy authorisation issues in a reliable and easy to configure way.
- The peer-based infrastructure of IOIDS and its employment of public-key-infrastructure technologies makes it resistant against a common set of well-known attack pattern.
- Decisions on data distribution and data integration are entirely left to the local node and are based on easy to configure rules, which have to be defined in XML encoded policy files.
- The completeness of components of the approach ensures consistence and integrity of data; meaning that data for the same event is always represented in exactly the same way in two different locations on the one hand and that no single piece of data is existent twice in the same form on the other hand.

Moreover, the following conclusions can be drawn from direct comparison of IOIDS with related approaches:

- IOIDS introduces a very clear and sober concept for classifying knowledge and distribute it under certain protection rules.
- In the feature category intrusion detection the only approach besides IOIDS satisfying all requirements is Prelude IDS, majorly due to its framework approach with plenty of integration facilities as carried out for IOIDS as well. The remaining approaches lack in features either because of focussing on a single set of detection mechanisms or missing integration facilities.
- Not a single approach from the list of chosen representatives is able to satisfy all issues in the category distribution. As half of the approaches even IOIDS itself does not cope

with realtime issues. Major drawbacks of other approaches are the lack of flexibility for the arrangement of nodes and the lack of trust relationships in the end.

- In terms of security and availability the approaches differ significantly due to their nature as well as there age. Recent approaches do employ PKI technologies and can, consequently, cover many of the issues arising in this category. Further problems are often based on centralised approaches and lack of authorisation features.
- For the category extensibility IOIDS is again the only approach satisfying the entire list of issues. Unaddressed issues for the other approaches were lack of integration with central management systems or minor objectives like modularity and applicability in heterogeneous environments.
- The IOIDS infrastructure is not prepared to hit the requirements regarding real-time issues. Several reasons for this drawback have been presented within this chapter; it became clear that certain components of the IOIDS approach need redesign for coping with these issues.

Issues around the intrusion detection focus such as further correlation and deep analysis of event data using data mining technologies or visualisation are not directly addressed by IOIDS; however, its integration with the SoapSy approach enables benefiting from all the research going on within the research group around this data repository.

After all, there is no such approach available, taking trust issues seriously into account. All the compared approaches are more or less focussing on data integration and data normalisation issues instead. This work has not yet been addressed by any approach out there to date.

7.5. Conclusion

In the introduction of this thesis the hypothesis has been laid down as statement that intrusion detection audit data may be exchanged across organisational boundaries in a secure and non-reputable manner, while maintaining commercial confidentiality. The objectives for such an approach have been discussed in very detail in chapter 3. Following this the definition of the experiment provided a first overview about the approach that has been put into place and, as well, about how an experiment can be able to prove the given hypothesis. After all the technical information about the major components G4DS and IOIDS in chapters 5 and 6, this chapter has discussed in very detail, how the before defined experiment has been executed and what conclusions could be drawn from the results.

The results combined from the feature comparison on the one hand and the practical experiments in the laboratory environment on the other hand have shown that IOIDS is capable of satisfying all the predefined objectives excluding the realtime-issue, which has been discussed in very detail within this chapter. Not only it was shown that IOIDS is able to fullfill the complete list of requirements, but also that it distinguishes itself from up-to-date approaches within the distributed intrusion detection system context by showing that each of them is lacking several of the defined features and, consequently, IOIDS can clearly make a contribution with its novel approach of knowledge sharing based on trust-relationships.

After the analysis of all the results and the mirroring against the requirements from the experiment definition the IOIDS system has finally been discussed in its design, architecture, implementation and usefulness. An overall conclusion in the upcoming chapter is summarising the information from all the available chapters and puts the achieved results back into the overall context of firstly distributed intrusion detection, and secondly information security in general.

Chapter 8.

Conclusions

The Inter-Organisational Intrusion Detection System infrastructure, as presented in this thesis, has proved that the exchange of intrusion detection audit data between organisations may be carried out in a secure and reliable manner while maintaining trust and commercial confidentiality.

A literature review has clearly presented the state of the art in intrusion detection technology and could identify the lack of such an approach in toady's information security environment. It could also be shown, that there is a requirement for the exchange of intrusion detection audit data on a high level across organisational boundaries as the currently available approaches require a significant amount of manual intervention and are, this way, not adequate to challenge nowadays threads in the information society.

The requirements and aims have been researched for IOIDS in the first place and a lot of knowledge from related or similar approaches within the DIDS as well as grid context could be reused. Objectives have been defined in detail in order to draw a picture of the desired project outcome.

8.1. Achievements

Objectives were broken down into low-level issues and, this way, an experiment could be defined, which basically described the way the IOIDS project could be evaluated against the aforementioned objectives. The outcome of this process was the following results:

- A detailed analysis plan was created, containing both a theoretical analysis using documentation of similar approaches and a practical part to be carried out in a laboratory environment. Both were made to examine the behaviour in comparison to other projects and to analyse, whether the previous defined objectives could be met.

- A high-level overview of the design, consisting of the four major components: Grid for

Digital Security (G4DS), Inter-Organisational Intrusion Detection System (IOIDS) application, Intrusion Detection System Integration Modules (IDS-IM) and the connected third-party Intrusion Detection Systems.

- A complete documentation for the design, architecture and implementation for the subjacent communication platform Grid for Digital Security (G4DS).

- A complete documentation for the design, architecture and implementation for the Inter-Organisational Intrusion Detection System application including all information for integration with third-party event generators such as intrusion detection systems, in detail for the network intrusion detection system Snort.

- A detailed documentation of the execution of the experiment, containing both:

  - The description of the comparison process for the theoretical experiment including the introduction of all representatives and the broken-down features. The attributes of each representative were mirrored against the set of requirements and a comparison was carried out in the end by using these results.

  - The execution of the practical experiment has been documented in two levels of detail; on the one hand an overview is given with results as part of the analysis process, on the other hand a detailed protocol is provided in the appendix in order to provide all technical information in order to rerun the experiments. Results have been documented whenever appropriate. The practical experiment was broken down into three stages in order to allow evaluation of the different components and structure the evaluation process.

With the analysis of the results in chapter 7 it could be shown that the Inter-Organisational Intrusion Detection System approach is superior to all available approaches concerning the requirements laid down for the project. The combination of the secure and reliable communication platform G4DS and the highly configurable and integrable Inter-Organisational Intrusion Detection System application provides an architecture for exchange of intrusion detection system audit data, which can be trusted by the parties because all the responsibility is strictly kept to the local node. Strong authentication and authorisation mechanisms ensure the security of information including all its features namely confidentiality, integrity and non-repudiation. Detailed results of the analysis process are presented in section 7.4.

8.1.1. Contribution to Science

There has not been published any approach or idea to date, which is addressing the exchange of intrusion detection audit data across organisational boundaries in a secure and reliable manner with putting major focus on the trust relationships between the involved parties. Loads of efforts have been put on structuring, generalising and normalising the audit data - the process of sharing, however, has not been addressed yet.

In detail, the following points describe the contribution in more detail:

- A peer-to-peer based communication infrastructure secured by use of public key infrastructure technology provides the base for protection of data.
- The introduction of trust relationships between parties and their implementation by use of trusting communities allows members to map real-world relationships into the system.
- Strict distribution of all functionality with total avoidance of single points of failure keeps the responsibility on the local node and, this way, enables members to trust the system.
- Implementation of plug-in mechanisms for third-party event generators opens the system for integration with any kind of audit data generating applications.

The academic society has also acknowledged the novelty of the Inter-Organisational Intrusion Detection System. Each of the following documents contains partial outcomes of the project; they have been peer-reviewed by the academic society and published in either journals or conference proceedings during the project lifetime:

- *Inter-Organisational Intrusion Detection using Knowledge Grid Technology*, Michael Pilgermann, Andrew Blyth, Stilianos Vidalis; Journal of Information Management and Computer Security, Volume 14, Number 4, 2006, ISSN: 0968-5227
- *Security in Heterogeneous Large Scale Environments Using GRID Technology*, Michael Pilgermann, Stilianos Vidalis, Evangelos Morakis, Andrew Blyth; International Journal for Innovative Computing, Information and Control (IJICIC), Volume 1, Number 4, December 2005; ISSN 1349-4198
- *GRID for Digital Security (G4DS)*, Stilianos Vidalis, Michael Pilgermann, Evangelos Morakis, Andrew Blyth; Journal of Maintenance Problems, 2(56), Instytut Technologii Eksploatacji - Panstwowy Instytut Badawczy, Poland, 2005, ISSN 1232-9312

- *Anonymizing data in a Peer-To-Peer based Distributed Intrusion Detection System*, Michael Pilgermann, Andrew Blyth; Paper for the ECIW 2004: The 3rd European conference on information warfare and security, 2004, ISBN: 0-9547096-2-4

8.2. IOIDS in information security

IOIDS can contribute significantly to the overall information security infrastructure. As it is not coming as an isolated solution but rather as a feature sitting on top of existing intrusion detection system facilities, it can integrate very well with available deployments. The additional knowledge, it can gain from other sides, enables the humans as well as the technology to improve and accelerate decision making based on a wider view of information.

8.3. Limitations and future work

As presented in the analysis of the experiment results (see section 7.4), the Inter-Organisational Intrusion Detection System was able to address the entire set of requirements but the real-time issue. However, during the project lifetime certain issues were raised, which could carry on the project or establish new related projects.

The following list presents a collection of potential projects that could take the outcome of the IOIDS project further:

- As mentioned several times, IOIDS with its proof-of-concept design and implementation was not able to address the real-time objective. Further research could investigate the bottle necks of the architecture and apply changes to design and implementation in order to overcome these problems. There might also be the option that wide-scale distribution of this kind of information cannot be carried out in realtime by nature due to the involved correlation, distribution and integration processes.

- In the current design, access control for G4DS is only implemented in two dimensions; meaning that it is worked out by the access control engine what actions which member is allowed on which targets by processing the rules. A perfectly sober design should extend that to the question of what member currently assigned to which community is allowed to carry out which operation on which target.

- The concept of anonymising and sanitising has been introduced in the chapter for Grid for Digital Security (chapter 5). The design and access control is prepared to handle these features in general; the architecture and implementation, however, do not include these features in the current stage.

- In the current stage there is a need to implement an IOIDS extension for every third-party application that shall be integrated with IOIDS; mainly in order to understand the extension information for this third-party event generator within the SoapSy database. In order to operate the SoapSy database, however, an XML document has to be provided, describing exactly this extension information. Theoretically, it should be possible for IOIDS to pick up all the information from this XML description and integrate new third-party applications dynamically.
- The Inter-Organisational Intrusion Detection System has only been deployed in the laboratory environment for the experiments for this project. Interesting results could be gained from wide-scale and longer-term deployments of IOIDS components, which share this information over the Internet instead of the simulation within a local area network.

Appendix A.

References

R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. Voms: an authorization system for virtual organizations. In *1st European Across Grids Conference*, Santiago de Compostela, 2003.

J. P. Anderson. Computer security threat monitoring and surveillance, 1980.

G. Angelis, S. Gritzalis, and C. Lambrinoudakis. Adressing authentication and authorization issues in the grid. Technical report, University of the Aegon, Department of Information and Communications Systems Engineering, 2001.

B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon. Web services security (ws-security) - version 1.0, April 2002.

N. Avourdiadis and A. Blyth. Soapsy - unifying security data from various heterogeneous distribute systems into a single database architecture. *The Journal of Information System Security (JISSec)*, 1(2), 2005a.

N. Avourdiadis and A. Blyth. Data unification and data fusion of intrusion detection logs in a network centric environment. In A. Blyth, editor, *Proceedings of the First European Conference on Network Defence (EC2ND)*, volume 1, December 2005b.

J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. *ACSAC*, 1998.

J. Barrus and N. C. Rowe. A distributed autonomous-agent network-intrusion detection and response system. In *Command and Control Research and Technology Symposium*, Monterey CA, June-July 1998.

K. Begain, G. Bolch, and H. Herold. *Practical Performance Modeling. Application of the MOSEL Language*. Kluwer Academic Publishers, 2001.

D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Tech. report mtr-2997, The MITRE Corporation, Bedford, MA, July 1975.

S. Bellovin, J. Schiller, and C. Kaufman. Rfc 3631: Security mechanisms for the internet, December 2003.

A. Blyth. An xml-based architecture to perform data integration and data unification in vulnerability assessments. *Information Security*, 8(4):14–25, 2003.

D. Box, G. Kakivaya, A. Layman, S. Thatte, and D. Winer. Soap: Simple object access protocol, 2000.

D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, M. Hadley, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, E. Sindambiwe, T. Storey, S. Weerawarana, and S. Winkler. Web services addressing (ws-addressing), August 2004.

J. Brentano, S. R. Snapp, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-l. Ho, K. N. Levitt, B. Mukherjee, and S. E. Smaha. An architecture for a distributed intrusion detection system. In *14th Department of Energy - Computer Security Group Conference*, pages 25–45, Concord, California, 1991.

D. F. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium On Research In Security And Privacy*, pages 206–214, OAKLAND, CALIFORNIA, 1989.

P. Brittenham. An overview of the web services inspection language, June 2002.

M. Bubak, P. Nowakowski, and R. Pajak. An overview of european grid projects. In *European Across Grids Conference 2003*, pages 299–308, 2003.

M. Cannataro and D. Talia. The knowledge grid. *Communications of the ACM*, 46(1):89–93, 2003.

C. C. Center. Snort xml output plugin - simple network markup language (snml). Website, 2003. URL `http://www.cert.org/kb/snortxml/`.

CERIAS. Autonomous agents for intrusion detection. Website, 2000. URL `http://www.cerias.purdue.edu/about/history/coast/projects/aafid.php`.

Cert/CC. Aircert. Website, 2006. URL `http://aircert.sourceforge.net/`.

Cert/CC. Cert / coordination center. Website, 2004. URL `http://www.cert.org/`.

D. Chadwick. An x.509 role based privilege management infrastructure. Technical report, October 2001, ISBN: 1-903150-52-3.

S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. Resource management in legion. *Future Generation Computer Systems*, 15(5-6):583 – 594, 1999.

D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In R. Rivest, editor, *Communications of the ACM*, 1981.

S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford-Chen, R. Yip, and D. Zerkle. The design of grids: A graph-based intrusion detection system, 1999.

R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. Internet Draft, March 2006. URL `http://www.w3.org/TR/2006/CR-wsdl20-20060327`.

E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1, March 2001.

I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with freenet, 2002.

L. Cottrell. Mixmaster & remailer attacks, 1995.

M. Crosbie and G. Spafford. Defending a computer system using autonomous agents. In *8th National Information Systems Security Conference*, 1994.

M. Crosbie and G. Spafford. Active defense of a computer system using autonomous agents, 15. Feb. 1995 1995.

D. Curry, H. Debar, and M. Lynch. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition, 2002.

CVE. Common vulnerabilities and exposes (http://www.cve.mitre.org), 2004.

K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, pages 181–193, 2001.

K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evaluation, February 2004a.

K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The ws-resource framework, March 2004b.

T. E. Daniels. A functional reference model of passive systems for tracing network traffic. *Digital Investigation - The International Journal of Digital Forensics & Incident Response*, 1:69–81, 2003.

R. Danyliw. Analysis console for intrusion databases. Website, 2006. URL `http://acidlab.sourceforge.net/`.

R. Danyliw, S. Levy, B. Trammell, and A. Kompanek. Aircert: The definitive guide, 2003.

R. Danyliw, J. Meijer, and Y. Demchenko. The incident object description exchange format data model and xml implementation. Internet-Draft, June 2006. URL `http://www.ietf.org/internet-drafts/draft-ietf-inch-iodef-07.txt`.

D. Davies, E. Holler, E. Jensen, S. Kimbleton, B. Lampson, G. LeLann, K. Thurber, and R. Watson. *Distributed Systems-Architecture and Implementation: An Advanced Course*. Lecture Notes in Computer Science. Springer Verlag Berlin / Heidelberg / New York, New York, 1981.

H. Debar, D. Curry, and B. Feinstein. The intrusion detection message exchange format (idmef). Internet-Draft, January 2005. URL `http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt`.

H. Debar, D. Curry, and B. Feinstein. The intrusion detection message exchange format draft-ietf-idwg-idmef-xml-16. Draft for RFC, 2006.

Y. Demchenko. Incident object description and exchange format requirements, 2003.

D. Denning. An intrusion detection model. *IEEE Trans. on Software Engineering*, SE-13(2), 1987.

T. Dierks and C. Allen. Rfc 2246: The tls protocol version 1.0, 1999.

T. Dierks and E. Rescorla. Rfc 4346 - the transport layer security (tls) protocol, version 1.1. Request for Comments, April 2006. URL `http://www.faqs.org/rfcs/rfc4346.html`.

W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

M. Dikaiakos, Y. Ioannidis, and R. Sakellariou. Search engines for the grid: A research agenda. In *1st European Across Grids Conference*, Universidad de Santiago de Compostela, Spain, 2003.

C. U. Dnad. On the feasibility of distributed intrusion detection. White Paper, 2004.

R. Dobson. Doing xml with t-sql. Internet, July 2004. URL `http://msdn.microsoft.com/xml/default.aspx?pull=/library/en-us/dnsqlpro04/html/sp04g1.asp`.

J. R. Douceur. The sybil attack. In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, 2001. Microsoft Research.

R. Favali and S. Stuart. Grid and web services standards to converge, 2004.

B. Feinstein, G. Matthews, and J. White. The intrusion detection exchange protocol (idxp), October 2002.

G. Fellows. Peer-to-peer networking issues - an overview. *Digital Investigation - The International Journal of Digital Forensics & Incident Response*, 1:3–6, 2004.

I. Foster. What is the grid? a three point checklist, 2003.

I. Foster, C. Kesselmann, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *5th ACM Conference on Computer and Communications Security Conference*, pages 82–92, 1998.

I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations, 2001.

I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

I. Foster, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savva, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The open grid services architecture, version 1.0, July 2004a.

I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling stateful resources with web services, March 2004b.

M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *ACM Conference on Computer and Communications Security (ACM CCS 9)*, Washington, DC, USA, 2002.

M. J. Freedman, E. Sit, J. Cates, and R. Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 1–6, MIT Faculty Club, Cambridge, MA, USA, 2001. MIT Laboratory for Computer Science.

Y. Fyodor. Snortnet - a distributed intrusion detection system, 2000.

D. Gannon, K. Chiu, M. Govindaraju, and A. Slominski. An analysis of the open grid services architecture, 2002.

I. Goldberg. *A Pseudonymous Communications Infrastructure for the Internet.* Phd thesis, UC Berkeley, 2000.

S. Goldwasser. New directions in cryptography: twenty some years later. *Proc. of 38th FOCS*, pages 314–324, 1997.

P. Golle and M. Jakobsson. Reusable anonymous return channels. In *Workshop on Privacy in the Electronic Society (WPES 2003)*, Washington, DC, USA, 2003.

P. Graham, M. Heikkurinen, J. Nabrzyski, A. Oleksiak, M. Parsons, H. Stockinger, K. Stockinger, M. Stroiski, and J. Aglarz. Eu funded grid development in europe. In *2nd EUROPEAN ACROSS GRIDS CONFERENCE*, Nicosia, Cyprus, 2004.

GRIDSTART. Gridstart - technical newsletter - issue 1, November 2002.

GRIDSTART. Gridstart - technical newsletter - issue 2, March 2003.

J. R. Groff and P. N. Weinberg. *SQL: The Complete Reference.* McGraw-Hill, 1999.

D. Gupta, T. C. Buchheim, B. S. Feinstein, G. A. Matthews, and R. A. Pollock. Iap: Intrusion alert protocol, 2001.

S.-J. Han and S.-B. Cho. Detecting intrusion with rule-based integration of multiple models. *Computers & Security*, 22(7):613–623, 2003.

L. T. Heberlein, B. Mukherjee, K. Levitt, and G. Dias. Towards detecting intrusions in a networked environment. In *14th Department of Energy Computer Security Group Conference*, 1991.

L. T. Heberlein, B. Mukherjee, and K. N. Levitt. Internetwork security monitor: An intrusion-detection for large-scale networks. In *15th National Computer Security Conference*, volume 1, pages 262–271, Baltimore, MD, 1992.

T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wobler. A network security monitor. In *IEEE Symposium on Research in Computer Security and Privacy*, pages 296–304, 1990.

C. L. Hedrick. Rfc 1058: Routing information protocol, 1988.

P. Hodson. *Local Area Networks*. Continium, London, 2003.

InvisibleNet. Invisible internet project (i2p), August 2003.

IT-ISAC. It information sharing and analysis center (it-isac), 2005.

P. Keleher, B. Bhattacharjee, and B. Silaghi. Are virtualized overlay networks too much of a good thing? In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, 2002.

R. Kolluru and P. H. Meredith. Security and trust management in supply chains. *Information Management & Computer Security*, 9(5):233 – 236, 2001.

W. Lee, R. A. Nimbalkar, K. K. Yee, S. B. Patil, P. H. Desai, T. T. Tran, and S. J. Stolfo. A data mining and cidf based approach for detecting novel and distributed intrusions. In H. Debar, L. Me, and S. F. Wu, editors, *Recent Advantages in Intrusion Detection*, pages 49–65, Toulouse, France, 2000. Department of Computer Science (North Caroline State University), Department of Computer Science (Columbia University).

D. Liesen. Requirements for enterprise-wide scaling intrusion detection products. White Paper, June 2002.

C. Lonvick. Request for comments: 3164 - the bsd syslog protocol. Internet Draft, August 2001. URL `ftp://ftp.rfc-editor.org/in-%notes/rfc3164.txt`.

Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *16th international conference on Supercomputing*, Columbia University, New York City, USA, 2002.

J. Mana. *Investigate Data Mining for Security and Criminal Detection*. 2003.

J. McLean. A comment on the 'basic security theorem' of bell and lapadula. *Information Processing Letters*, 20(2):67–70, 1985.

R. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21 (4):294–299, 1978.

S. Micali. Fair public-key cryptosystem. *Advances in Cryptology, Proc. of Crypto '92*, LNCS 740:113–138, 1992.

E. Miller. An introduction to the resource description framework. *D-Lib Magazine*, May 1998.

D. Moore and J. Hebeler. *Peer-to-Peer - Building Secure, Scalable and Manageable Networks.* McGraw-Hill, Osborne, 2002.

E. Morakis, S. Vidalis, and A. Blyth. A framework for representing and analysing cyber attacks using object oriented hierarchy trees. In *The 2nd European Conference On Information Warfare And Security (ECIW)*, Reading, UK, 2003.

B. Mukherjee, L. Heberlein, and K. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, 1994.

N. Nagaratnam, P. Janson, J. Dayka, A. Nadalin, F. Siebenlist, V. Welch, I. Foster, and S. Tuecke. The security architecture for open grid services, July 2002.

Netfilter. Netfilter / iptables. Website, 2005. URL `http://www.netfilter.org/`.

Netscape. Ssl 3.0 specification, November 1996.

E. Newcomer. *Understanding Web Services- XML, WSDL, SOAP and UDDI.* Independant Technology Guides, 1 edition, May 2002.

NSS. Gigabit intrusion detection systems - group test edition 1. Technical report, NSS Group, December 2002.

NSS. Public key infrastructure (pki) - group test (edition 6). Group test, NSS Group, 2004.

NSS. Internet security - "the modern day gold rush", 2005.

W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *Information and Communications Security — First International Conference*, pages 440–444, 1997.

V. Parmar, H. Shi, and S.-S. Chen. Xml access control for semantically related xml documents. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, volume 9, page 288.2. IEEE Computer Society Washington, DC, USA, 2003.

A. Pfitzmann and M. Waidner. Networks without user observability. *Computers and Security*, 6(2):158–166, 1987.

C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*. Prentice Hall Professional Technical Reference, New Jersey, 3 edition, 2003.

M. Pilgermann. *Absicherung von Netzen mit Hilfe von Intrusion Detection / Intrusion Prevention Systemen - Resistenz gegen Anti-IDS-Angriffe*. diploma thesis, University of Applied Scienes and Arts Hannover, 2003.

M. Pilgermann and A. Blyth. Anonymizing data in a peer-to-peer based distributed intrusion detection system - a possible approach. In A. Jones and D. Remenyi, editors, *European Conference on Information Warfare (ECIW) 2004*, London, 2004, ISBN: 0-9547096-2-4.

D. M. Piscitello and A. L. Chapin. *Open Systems Networking: TCP/IP and OSI*. Addison-Wesley Longman Publishing Co., Inc., 1993.

S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theor*, IT-24(1): 106–110, 1978.

J. Postel. Domain name system structure and delegation. RFC, March 1994. URL `http://www.isi.edu/in-%notes/rfc1591.txt`.

Prelude. Prelude-ids - the hybrid ids framework. Website. URL `http://www.prelude-ids.org/`.

L. Quin. *Open Source XML Databases Toolkit - Resources and Techniques for Improved Development*. Wiley Computer Publishing, 2000.

X. Quin and W. Lee. Statistical causality analysis of infosec alert data. In *Recent Advances in Intrusion Detection (RAID) 2003*, pages 73–93, Pittsburgh, PA, USA, 2003.

J. F. Rafael Paez, Cristina Satizabal. A virtual intrusion detection system (vids) for virtual organizations. *expected*, 2006.

E. T. Ray. *Learning XML*. O'Reilly, 1 edition, January 2001.

M. Rennhard, S. Rafaeli, and L. Mathy. The pseudonymity network architecture. Technical report, Swiss Federal Institute of Technology, Computer Engineering and Networks Laboratory; Zurich, Switzerland; Lancaster University, Faculty of Applied Sciences; Lancaster, UK, February 2001a.

M. Rennhard, S. Rafaeli, L. Mathy, B. Plattner, and D. Hutchison. An architecture for an anonymity network. In *IEEE 10th Intl. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2001)*, 2001b.

M. Y. Rhee. *Internet Security - Cryptographic principles, algorithms and protocols*. John Wiley & Sons Ltd, The Atrium, Chichester, West Sussex, England, 2003.

R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems and public-key cryptosystems. *Communications of the ACM*, 21 (2):120–126, 1978.

M. Rose. The blocks extensible exchange protocol core, 2001.

K. Scribner and M. C. Stiver. *Understanding SOAP*. Sams Publishing, 1 edition, 2000.

S. Seely. *SOAP - Cross Plattform Web Service Development Using XML*. Prentice-Hall, New Jersey, USA, 2003.

A. Serjantov. Anonymizing censorship resistant systems. In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, 2002. University of Cambridge Computer Laboratory.

SETI. Seti at home, 2005. URL `http://setiathome.berkeley.edu/`.

A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

S. Snapp, J. Brentano, G. Dias, T. Goan, L. Heberlein, C. Ho, K. Levitt, and B. Mukherjee. A system for distributed intrusion detection. In *COMPCON Spring '91*, pages 170–176, San Francisco, 1991a.

S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. Dids (distributed intrusion detection system) motivation, architecture, and an early prototype. In *14th National Computer Security Conference*, volume 1, pages 167–176, Washington, D.C., 1991b.

Snort. Snort - the de-facto standard for intrusion detection. Internet site, 2006. URL `http://www.snort.org/`.

E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):547 – 570, October 2000. URL `http://portal.acm.org/citation.cfm?id=361120`.

S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids - a graph-based intrusion detection system for large networks. In *19th National Information Systems Security Conference*, 1996.

W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

M. Strembeck. A role engineering tool for role-based access control. In *Proc. of the Symposium on Requirements Engineering for Information Security (SREIS)*, Paris, France, August 2005.

D. Talia. Enabling knowledge discovery services on grids. In *2nd EUROPEAN ACROSS GRIDS CONFERENCE*, Nicosia, Cyprus, 2004.

ThePreludeTeam. Prelude 0.9 handbook. Documentation, 2006. URL `https://trac.prelude-ids.org/wiki/PreludeHandbook`.

M. Tolba, I. Taha, and A. A. Shishtawy. An intrusion detection architecture for computational grids. In *First International Conference ICICIS*, June 2002.

M. Tolba, M. Abdel-Wahab, I. Taha, and A. Al-Shishtawy. Distributed intrusion detection system for computational grids. In *Second International Conference on Intelligent Computing and Information Systems*, March 2005a.

M. F. Tolba, M. S. Abdel-Wahab, I. A. Taha, and A. M. Al-Shishtawy. Gida: Toward enabling grid intrusion detection systems. In *5th IEEE International Symposium on Cluster Computing and the Grid*, 2005b.

B. Trammell, R. Danyliw, S. Levy, and A. Kompanek. Aircert: The definitive guide. Documentation, 2005. URL `http://aircert.sourceforge.net/docs/aircert_manual-06_2005.pdf`.

B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Pouyoul, and B. Yeager. Project jxta 2.0 super-peer virtual network, 2003.

TruSecure. Icsa labs ids consortium announces network intrusion detection system alert specification format. Press Release, February 2004. URL `http://www.trusecure.com/company/press/pr_20040223.shtml`.

S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open grid services infrastructure - version 1.0, June 2003.

Y. Vandoorselaere, L. Oudot, and M. Schillinger. Prelude: an open source, hybrid intrusion detection system - architecture guide, 2004.

S. Vidalis, E. Morakis, and A. Blyth. Security in heterogeneous large scale environments using grid technology. Technical report, University of Glamorgan - School of Computing, 2003.

W3C. Xml schema specifications. Internet drafts, 2005. URL `http://www.w3.org/XML/Schema.html#dev`.

W. W. W. C. (W3C). Document object model (dom). Internet site, 2005. URL `http://www.w3.org/DOM/`.

M. Waidner and B. Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. *Advances in Cryptology EUROCRYPT 89, LNCS434*, 1990.

D. S. Wallach. A survey of peer-to-peer security issues. 2001.

B. R. Waters, E. W. Felten, and A. Sahai. Receiver anonymity via incomparable public keys. In *ACM Conference on Computer and Communications Security*, pages 112 – 121, Washington D.C., USA, 2003.

V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, June 2003. URL `http://www.globus.org/alliance/publications/papers/GT3-Security-HPDC.pdf`.

G. White, E. Fisch, and U. Pooch. Cooperating security managers: a peer-based intrusion detection system. *Network, IEEE*, 10(1):0890–8044, 1996.

Wikipedia. Bell-lapadula model. Internet Page, July 2006. URL `http://en.wikipedia.org/wiki/Bell-LaPadula`.

G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume 2 - The Implementation*, volume 2. Addison-Wesley Publishing Company, 1995.

K. Xynos. A soap-xml interface which will interpret sql commands for an ids database. Master's thesis, University of Glamorgan, September 2005.

Appendix B.

Details for experiment execution

B.1. Execution Protocols

Within this section, detailed information is provided about the way, the experiments have been carried out. I try to break down the whole process into single steps in order to enable everybody to rerun the experiments. Due to the vast amount of log information, it was impossible to attach them all to this document. Instead, a CD was created, which holds all this information. References into this CD are made whenever appropriate.

The protocols have been ordered by the dates, they were carried out. Whereever useful, the times have been provided as well; usually in order to simplify searching within log files for certain time frames.

B.1.1. Stage 1

This subsection describes the steps, which have been carried out for experiment execution stage 1 as defined in section 4.4.1. The analysis of this information is maintained in section 7.3.1 of the analysis chapter.

Firstly, the process of setting up the laboratory environment is shown. Afterwards, attention is drawn to the execution of the stage one experiment itself; broken down into its 4 steps, namely:

1. Correct messages
2. Faulty distribution domains
3. Access Control evaluation
4. Penetration

Preparation

The following sections are a detailed collection of steps, which have been executed for the experiment in the laboratory environment.

15/05/2006 – Setup of network nodes

- Setup of 6 personal computers with software as described in tables B.7 and B.8
- Applying of updates on those machines
- Installation of Python with required libraries (site-packages)
- Installation of PostgreSQL database management system on each machine
- Creation and configuration of database for g4ds use (details are shown in table 7.5)
- Apply SQL script *create_tables.sql* on the newly created database for generating required relations for the g4ds system.

16/05/2006 – Installation of g4ds application and its configuration

- Configuration of each machine by following the following steps:
  1. Unpack g4ds archive *g4ds-0.1.tar.gz*
  2. Edit g4ds configuration file *g4ds.conf* and apply information for member id, IP address information and database connectivity information (details are provided in table 7.6).
  3. (Only for M001 and M002): remove the own entry from list of member descriptions to be loaded on installation start since the node itself is an authority
  4. (Only for non Gentoo installations): Copy the python library *output.py* from the *thirdparty* directory to the g4ds directory.
  5. Distribute g4ds to the local system by running the python setup: *python setup.py install*
  6. Run the g4ds installation (distribute initial knowledge to the databases) by executing *python install.py* from the g4ds directory (as a sample, a copy of the corresponding output for running the given command on node M004 is provided in Listing B.15).
  7. Export the local member description to a file.
- After M001 and M002 had been created, the community description for the default community *C9999999999* could be manually updated and applied to all other nodes.
- All member descriptions from all nodes have been applied to all other nodes.
- Configuration of communities and applying descriptions to the corresponding nodes by performing the following steps (details for community configurations are given in table 7.7):
  1. Creation of community description file. (As a sample a copy of the community description for community *C001* is provided in listing B.16)
  2. Apply community description to community authorities using g4ds maintain environment. (In the first turn apply without gateway information as the linked community is not yet known on the local node. Only if the link community is created first, the gateways may be processed.)
  3. Apply community description to remaining members using g4ds maintain environment.
  4. Subscribe the remaining members to the community using g4ds maintain environment by sending a request to one of the authorities.
  5. Recalculate routing table for all members using maintain environemnt. ‘
  6. Generate endpoints for the new community for each member using maintain environment.
  7. Generate, export and import member descriptions for all members using maintain environment.
- Configure and apply chat service on all nodes. (A copy of the source code for the program is provided in listing B.17. A service description had to be created - a

copy of this may be found in listing B.18.) For each node the following steps had to be gone through:

1. A private/public key pair had to be created using the g4ds maintainence environment.
2. The Service Description (SDL) file had to be applied using the g4ds maintainence environment for making the service known to G4DS.
3. The generated key had to be made accessable to the chat service program.

17/05/2006 – Finish of service installation, synchronise machines and start experiments

- The nodes had to subscribe to the service using the g4ds maintainence environment (option *2*).
  As nodes M004 - M006 are located behind an NAT (M003), it is impossible to establish a direct connection from M001 or M002 (which act as service authorities for this service) to inform these nodes about the sucess of their subscription (This will not be an issue later on as all information for those nodes will only be distributed within community C003, saying within the NATed network). However, manual entries had to be added to the g4ds repository in order to map the subscription. As an example, this comamnd for node M004 is:

Listing B.1: Manual adjustment of g4ds database

```
echo ``insert into services_members values ('S123456','M004')'' | psql -
    U ug4ds -h grid01 g4ds
```

- In order to simplify tracking of messages throughout the laboratory environment (e.g. by tracing logging information) it was necessary to synchronise the times of the machines. The following configuration was set up:
  - Node M003 (193.63.129.193) acts as the central time server. It uses its own hardware clock as the only time reference - no internet connection is necessary. (In the end it was only important to keep the machines synchronised between each other).
  - The remaining nodes (M001, M002, M004 - M006) are acting as clients and receive their time information using pull mechanism frequently from M003. Besides the local time as a fall-over source no other time source is used.

Stage 1 - Step 01 - Correct messages

In the first part of the stage 1 experiments, ordinary messages were populated around the network in order to examine the abilities of G4DS to distribute information. The following messages were created using the aforementioned chat service.

17/05/2006 - Start correct messages

- Populate messages from node M001

  - The following actions were carried out for this step:
    1. Start G4DS service on all nodes
    2. Start chat application on all nodes
    3. Send a messages to the list of receivers
    4. Shut down the chat application
    5. Shut down G4DS service
    6. Move log files to output folder
  - Distribution list: M001, M002, M003, M006, C001, C002, C003
  - Time frame for this step: **19.47 – 19.55**
  - All logging information available in *output/$NODE/stage1/001/a* (g4ds logging and console output from chat service)

18/05/2006 - Correct messages continued

- Populate messages from node M002
  - The actions are exactly the same as described before for M001
  - Distribution list: M001, M003, M005, C001, C002, C003
  - Time frame for this step: **10.55 – 11.03**
  - All logging information available in *output/$NODE/stage1/001/b* (g4ds logging and console output from chat service)
- Populate messages from node M003
  - The actions are exactly the same as described before for M001
  - Distribution list: M002, M006, C001, C002, C003
  - Time frame for this step: **11.32 – 11.38**
  - All logging information available in *output/$NODE/stage1/001/c* (g4ds logging and console output from chat service)
- Populate messages from node M004
  - The actions are exactly the same as described before for M001
  - Distribution list: M002, M001, M003, M005, C001, C002, C003
  - Time frame for this step: **11.40 – 11.46**
  - All logging information available in *output/$NODE/stage1/001/d* (g4ds logging and console output from chat service)
- Populate messages from node M005
  - The actions are exactly the same as described before for M001
  - Distribution list: M002
  - Time frame for this step: **11.48 – 11.52**
  - All logging information available in *output/$NODE/stage1/001/e* (g4ds logging and console output from chat service)
- Populate messages from node M006

  - The actions are exactly the same as described before for M001
  - Distribution list: M002
  - Time frame for this step: **11.53 – 11.57**
  - All logging information available in *output/$NODE/stage1/001/f* (g4ds logging and console output from chat service)

Stage 1 - Step 02 - Faulty distribution domains

In the second step of the first stage distribution was facing two problems, either the distribution domain is faulty for the local node or one one within the distribution domain is down. The following messages were sent using the chat service in order to examine g4ds‘ reaction to these problems:

18/05/2006 – Faulty distribution

- Faulty distribtion from M002 - unknown distribution domains
  - The following actions were carried out for this step:
    1. Start G4DS service on all nodes
    2. Start chat application on all nodes
    3. Send a messages to the list of receivers
    4. Shut down the chat application
    5. Shut down G4DS service
    6. Move log files to output folder
  - Distribution list: M002, C007
  - Time frame for this step: **12.16 – 12.21**
  - All logging information available in *output/$NODE/stage1/002/a* (g4ds logging and console output from chat service)
- Faulty distribtion from M003 - unknown distribution domains
  - The actions are exactly the same as described before for M001
  - Distribution list: C007, M000
  - Time frame for this step: **12.25 – 12.29**
  - All logging information available in *output/$NODE/stage1/002/b* (g4ds logging and console output from chat service)
- Faulty distribtion from M002 - chat application on node M003 no started
  - The actions are exactly the same as described before for M001; however, the chat service is not started on node M003
  - Distribution list: M002, C002, M005
  - Time frame for this step: **12.30 – 12.34**
  - All logging information available in *output/$NODE/stage1/002/c* (g4ds logging and console output from chat service)

- Faulty distribtion from M004 - g4ds service not started on node M002
  - The actions are exactly the same as described before for M001; however, nor the g4ds service neither the chat service are started on node M002
  - Distribution list: M002, C001
  - Time frame for this step: **12.35 – 12.39**
  - All logging information available in *output/$NODE/stage1/002/d* (g4ds logging and console output from chat service)
- Faulty distribtion from M002 and M006 - g4ds service not started on node M003
  - The actions are exactly the same as described before for M001; however, nor the g4ds service neither the chat service are started on node M003
  - Distribution list: for M002 - C002, M006; for M006 - M003, M002
  - Time frame for this step: **12.42 – 12.46**
  - All logging information available in *output/$NODE/stage1/002/e* (g4ds logging and console output from chat service)

Stage 1 - Step 03 - Access control evaluation

The third step of the stage 1 experiment evaluates access control features of G4DS. For these reason, changes were applied to the G4DS access control policies before messages were populated. The following steps were carried out for this step in detail:

18/05/2006 – G4DS Access Control

- Backup access control ruleset on every node
- M002 blocks M005 application messages
  - The following actions were carried out for this step:
    1. Access control policy files have to be adjusted
    2. Start G4DS service on all nodes
    3. Start chat application on all nodes
    4. Send a messages to the list of receivers
    5. Shut down the chat application
    6. Shut down G4DS service
    7. Move log files to output folder
    8. Access control policy files have to be resetted
  - Access Control Policy changes: On node M002 all incoming application messages with sender M005 are dropped
  - Distribution list: M004 to M002; M005 to M002; M005 to C001
  - Time frame for this step: **13.34– 13.38**
  - All logging information available in *output/$NODE/stage1/003/a* (g4ds logging and console output from chat service)

- M001 blocks M003 application messages
  - The actions are exactly the same as described in the first case of this step
  - Access Control Policy changes: On node M001 all incoming application messages with sender M003 are dropped
  - Distribution list: M003 to M001; M003 to M002; M005 to M001
  - Time frame for this step: **17.17– 17.21**
  - All logging information available in *output/$NODE/stage1/003/b* (g4ds logging and console output from chat service)
- M002 blocks messages from community C002
  - The actions are exactly the same as described in the first case of this step
  - Access Control Policy changes: On node M001 all incoming messages sent within community C002 are dropped
  - Distribution list: M003 to M002; M005 to M002
  - Time frame for this step: **18.07– 18.11**
  - All logging information available in *output/$NODE/stage1/003/c* (g4ds logging and console output from chat service)
- M001 blocks application messages for chat service
  - The actions are exactly the same as described in the first case of this step
  - Access Control Policy changes: On node M001 all incoming application messages for service with S123456 are dropped
  - Distribution list: M005 to M001; M001 to M001; M005 to M002
  - Time frame for this step: **18.34– 18.37**
  - All logging information available in *output/$NODE/stage1/003/d* (g4ds logging and console output from chat service)

Stage 1 - Step 04 - Penetration

The last step of the first stage describes the way G4DS was penetrated in order to check its resistance against Sniffing, Spoofing and DOS attacks. For the spoofing attacks a small Python program was created in order to replace member ids within G4DS messages (program source code in listing B.20). The ethereal application was started with the following parameters in order to generate raw output:

Listing B.2: Call of Ethereal network sniffer

```
tethereal -i eth0 -f ``port not 22 and not arp and not broadcast and not
    multicast'' -w $outputFilename
```

For evaluation of the ethereal results, the Ethereal-X-application was started and the dump files were loaded into it.

Furthermore, there was a need to direct network traffic to tcp ports, g4ds is listening on for denial-of-service attacks and spoofing attacks. Using netcat, the compilation of the commands is shown in listing B.3.

Listing B.3: Commands for G4DS denial of service

```
# netcat commands for DOS
#
while test 1:
do
  nc j4-itrl -12 8080 < soap-dump.m001.txt >/dev/null &  # for soap or
  nc j4-itrl -12 2000 < tcp-dump.m001.txt >/dev/null &  # for tcp socket
  sleep 1   # if the delay between 2 packets is 1 second
  killall nc
done
#
# netcat command for spoofing
nc -g -s 193.63.129.193 193.63.129.184 2000 < tcp-dump.txt
```

In detail, the following actions were carried out for this step:

19/05/2006 – G4DS Penetration

- Sniffing on M001
  - Little changes were applied to the g4ds source code in order to simulate a node within the network, which acts as a gateway and attempts to read information, which it is supposed to passed on to another node.
  - The following actions were carried out for this step:
    1. Start G4DS service on all nodes
    2. Start chat application on all nodes
    3. Start Ethereal capturing
    4. Send a messages to the list of receivers from within this distribution list
    5. Stop Ethereal, save ethereal dump and go back to step 3 of this list until there is no more distribution lists
    6. Shut down the chat application
    7. Shut down G4DS service
    8. Move log files to output folder
  - Distribution domains and time frames of execution:
    1. 01: M002 to M001, M003, M005; Time frame: **17:35 - 17:39**
    2. 02: M001 to M001, M002, M003; Time frame: **17:40 - 17:44**
    3. 03: M003 to M003, M001, M002; Time frame: **17:45 - 17:48**
    4. 04: M005 to M001, M002, M003; Time frame: **17:48 - 17:52**
  - All logging information available in *output/$NODE/stage1/004/a{01-04}* (g4ds logging and console output from chat service)

20/05/2006 – G4DS Penetration continued

- Flood g4ds with netcat
  - The following actions were carried out for this step:

1. Start ethereal to capture sample traffic for a certain protocol, employed by g4ds
2. Temporarely uncomment endpoints for the destination of the sample messages for all protocols but the disired one within that endpoints member description and apply this updated description using G4DS maintainence environment
3. Generate sample traffic by sending one message between 2 nodes
4. Stop Ethereal and extract application data using GUI
5. If more protocols are to be captured, choose another one and go back to item 1
6. Reset and apply all member descriptions
7. Start G4DS service on all nodes
8. Start chat application on all node
9. Start background noise (penetration) using netcat as described in listing B.3 (DOS part)
10. Apply changes on the local g4ds implementation in order ot force g4ds to use a certain protocol (edit the configuration module *protocols/config.py*)
11. Send ordinary messages from list
12. Stop background noise
13. Shut down chat application
14. Shut down G4DS service
15. Move log files to output folder

– Penetration nodes and distribution domains:
  1. 01: Noise are g4ds inside soap requests from M002 to M001
     Distribution list: M002 to M003, M001; M005 to M003, M001; M003 to M003, M001
     Time frame: **13:23 - 13:29**
  2. 02: Noise are g4ds inside TCP/IP packets from M002 to M001
     Distribution list: M002 to M003, M001; M005 to M003, M001; M003 to M003, M001
     Time frame: **14:12 - 14:18**
  3. 03: Noise are soap requests from Ext to M001 (from external source)
     Distribution list: M003 to M003, M001; M005 to M003, M001
     Time frame: **14:49 - 14:54**
  4. 04: Noise are TCP/IP packets from Ext to M001 (from external source)
     Distribution list: M003 to M003, M001; M005 to M003, M001
     Time frame: **14:59 - 15:03**
– All logging information available in *output/$NODE/stage1/004/b{01-04}* (g4ds logging, console output from chat service, sample traffic)

22/05/2006 – G4DS Penetration continued

- Spoofing
  - The following actions were carried out for this step:
    1. As performed for the previous step (g4ds flooding) generate a sample message and capture it using tethereal; extract application data using Ethereal
    2. Modify the identity for the source member within the message using the program shown in listing B.20 and save the fakeMessage
    3. Start G4DS service on nodes M001, M002, M003
    4. Start chat application on nodes M001, M002, M003
    5. Send messages (plain or faked) to M001 (j4-itrl-12) using netcat from M002 (j130-mp) (see listing B.3 for detailed command) - see list of messages directions underneath
    6. Stop chat application on nodes M001, M002, M003
    7. Stop G4DS service on nodes M001, M002, M003
    8. Move log files to output folder
  - Messages and time frames of execution (all messages are sent from M002 to M001):
    1. 01: Normal message - no failure; Time frame: **15:47 - 15:55**
    2. 02: Spoofed IP address; Time frame: **15:58 - 16:02**
    3. 03: Spoofed G4DS member id; Time frame: **15:47 - 15:55**
    4. 02: Spoofed IP address and member id; Time frame: **15:58 - 16:02**
  - All logging information available in *output/$NODE/stage1/004/c{a-d}* (g4ds logging, console output from chat service)

B.1.2. Stage 2

This section provides detailed information about the actions carried out for stage 2 of the experiment in the laboratory environment. The definition of the experiment for this stage are written down in section 4.5.2; the corresponding analysis part may be found in section 7.3.2.

After providing information about the way, the laboratory environment has been modified in order to be suitable for carrying out the experiments for this stage, the details for this stage are presented on a one subsection per step basis. Stage two of the experiment has been broken down into the following five steps:

- Sending of correct messages
- Rejection of messages
- Insertion of remote events
- Content and different levels of details
- Performance and overhead measurements

Preparation

For carrying out the second stage of the experiment, the laboratory environment has been taken from the first stage and extended by certain components and features. The single steps for these changes are as follows.

06/06/2006 - Modifications for laboratory environment

- The Inter-Organisational Intrusion Detection System application itself had to be deployed on every node within the G4DS network. For this step the instructions given in the IOIDS installation guide have been used (see listing B.21). In detail, I went through the following list of items for each node in the G4DS network:
  - A PostgreSQL database management system (DBMS) has been running on each node already. For the IOIDS repository a new database had to be created and privileges had to be set up. Each node is using its own host for the database, username is *uioids*, as password the phrase *psoioids* was assigned. The database itself was named *ioids*. Privileges had to be set up in a way that the user uioids is allowed to connect to the database ioids using password authentication from the ip address of the node in question. (The PostgreSQL access control file *pg_hba.conf* had to be extended by one line.)
  - The python library SoapSyTools had to be installed. As part of the project I realised that certain parts of the IOIDS implementation are reuseable for other application, connecting against the SoapSy database. Consequently, I excluded this functionality into its own library. It has to be unpacked and installed as a Python site-package using the command:
    python setup.py install
  - Configuration of G4DS: The IOIDS service description had to be applied to G4DS using the G4DS maintainence environment. Furthermore, a service key had to be created and exported into a location accessable for the IOIDS application later on.
  - The IOIDS package has to be unpacked.
  - Configuration of XSM: XSM is the server-side process running on the SoapSy server in order to provided XML-based RPC access to the databae. XSM has not been developed as part of my project; however, I added a version into the IOIDS archive, which is working with IOIDS. It has to be configured in order to connect against the just installed IOIDS database on startup time. For these reasons, the configuration file *XSM-Configuration.xml*, located underneath the third-party folder was edited and settings as applied for the ioids database were applied.
  - The IOIDS service itself has to be configured using the configuration module *config.py* located in the root folder within the archive. Settings have to be applied for making IOIDS connected against XSM at startup time (the host therefore should be localhost by default). Furthermore, I specified the location of the G4DS key, which was created in one of the former steps.

– Install XSM and IOIDS on the system and distribute files in the local filesystem by running the following command in the ioids folder:
python setup.py install
– Afterwards, privileges had to be applied on the files */etc/init.d/xsmrc* and */usr/sbin/XSM.py* in order to make them executable for the user:
chmod 755 /etc/init.d/xsmrc && chmod 755 /usr/sbin/XSM.py The final step on each node was to integrate IOIDS with the G4DS access control. The available g4ds policy had to be extended by a rule in order to redirect all IOIDS messages access control requests to the new (provided) ioids policy file. Furthermore, this policy file had to be registered with G4DS by providing its name in the G4DS configuration file */etc/g4ds.conf* (see IOIDS installation instructions for more details).

07/06/2006 - Modifications for laboratory environment continued

- In order to finish of the IOIDS installation, the IOIDS date engine had to be configured:
  – The IOIDS data engine is configured using a policy file named *ioids_policy.xml*, which is maintained in the folder *descriptions* underneath the IOIDS folder. Each node was configured in a way that it generates local events for each community it is a member of. No messages from remote hosts are to be passed on. The distribution domains for each node are given in table B.1. (The trustees row in there specifies, from which nodes incoming information should be processed.)

Table B.1.: IOIDS DataEngine Distribution and Trustees

	M001	**M002**	**M003**	**M004**	**M005**	**M006**
C001	*	*				
C002	*		*			
C003			*	*	*	*
Trustees	M001	M001	M001	M003	M003	M003
	M002	M002	M003	M004	M004	M004
	M003		M006	M005	M005	M005
				M006	M006	M006

Stage 2 - Step 01 - Correct messages

Within this step a single event shall be inserted into each SoapSy database, which is to be picked up by the IOIDS system in order to send it off to its distribution domain(s).

07/06/2006 - Sucessful messages from each node

- On each node the following actions had to be performed:
  - Protocol the IOIDS database status (latest event ids)
  - Start the G4DS service on every node
  - Start the XSM service on every node
  - Start IOIDS on every node
  - For each node within the G4DS network
    - Insert one event manually into the SoapSy database using the soapClient coming with the XSM XML RPC program
    - Wait 2 minutes before progressing to the next node in order to maintain a time gap within the log files between the events
  - Stop IOIDS on every node
  - Stop the XSM service on every node
  - Stop the G4DS service on every node
  - Relocate available logging information into corresponding output folder
- Time frame for this step: **13.38 – 13.50**
- All logging information available in *output/$NODE/stage2/001* (g4ds logging, ioids logging, database status information and console output from manual event insertion)

Stage 2 - Step 02 - Rejected messages

In contrast to the first step of this stage this one shall examine features of IOIDS alone, firstly, and in correlation with G4DS, secondly, to filter certain messages, which are not supposed to be processed on the local node.

07/06/2006 - Filter IOIDS messages

- Filter messages using G4DS access control
  - Before the tests were run, the following changes to the access control policies had been applied on the following nodes:
    - M001: reject all writing access for service IOIDS from M002
    - M003: allow all writing access for service IOIDS from M002
  - Before the test were run, the following changes to the IOIDS data engine had been applied on the following nodes:
    - M002: Distribute to M003 (on top of the existing rule, defining distribution to C001)
  - Afterwards, the folling steps were gone through on each node:
    - Protocol the IOIDS database status (latest event ids)
    - Start the G4DS service on every node
    - Start the XSM service on every node
    - Start IOIDS on every node

    - Insert one event on each of the nodes (M003, M002) manually into the SoapSy database using the XSM RPC soap client and wait 2 minutes before progressing to the next node in order to maintain a time gap within the log files between the events
    - Stop IOIDS on every node
    - Stop the XSM service on every node
    - Stop the G4DS service on every node
    - Relocate available logging information into corresponding output folder
  - Note: As there was a problem with the data engine rule on node M003 in the first run (it was not accepting service messages from M002), the second part was run again.
  - Time frame for this step: **16.02 – 16.17**
  - All logging information available in *output/$NODE/stage2/002/a* (g4ds logging, ioids logging, database status information and console output from manual event insertion)
- Filter messages using IOIDS data engine policies
  - Before the tests were run, the following changes to the access control policies had been applied on the following nodes:
    - M002: allow all writing access for service IOIDS from source M003
    - M003: allow all writing access for service IOIDS from source M002
  - Afterwards, the folling steps were gone through on each node:
    - Protocol the IOIDS database status (latest event ids)
    - Start the G4DS service on every node
    - Start the XSM service on every node
    - Start IOIDS on every node
    - Insert one event on each of the nodes (M002, M003) manually into the SoapSy database using the XSM RPC soap client and wait 2 minutes before progressing to the next node in order to maintain a time gap within the log files between the events
    - Stop IOIDS on every node
    - Stop the XSM service on every node
    - Stop the G4DS service on every node
    - Relocate available logging information into corresponding output folder
  - Note: As there was a problem with the g4ds access control rule on node M002 in the first run (it was not allowing service messages from M003), the first part was run again.
  - Time frame for this step: **16.38 – 16.49**
  - All logging information available in *output/$NODE/stage2/002/b* (g4ds logging, ioids logging, database status information and console output from manual event insertion)

Stage 2 - Step 03 - Remote events

The IOIDS data engine is capable to pass on ioids events from remote sources to other nodes within the network. It shall be examined, whether information is passed on correctly, as indicated in the policy files, and whether information is kept cosistent.

08/06/2006 - Remote Events

- Pass on IOIDS events from remote sources to other nodes
  - Before the test were run, the following changes to the IOIDS data engine had been applied on the following nodes (listing B.4 shows one new date engine rules applied on M001 for these reasons):
    - M001: Any remote event from C002 shall be passed on to member M002 and any remote event from C001 shall be passed on to member M003
    - M003: Any remote event from either C001 or C002 shall be passed on to membes M004, M005 and M006
  - Afterwards, the folling steps were gone through:
    - Protocol the IOIDS database status (latest event ids) on every node
    - Start the G4DS service on every node
    - Start the XSM service on every node; the XSM service was kept running from now on and not shut down at all anymore during experiment execution
    - Start IOIDS on every node
    - Insert a new event manually into the SoapSy databases of the nodes given in the distribution list; leave 2 minutes time gap between each event in order to simplify tracing log files
    - Stop IOIDS on every node
    - Stop the G4DS service on every node
    - Relocate available logging information into corresponding output folder
  - Distribution list: M001, M002, M003, M004
  - Time frame for this step: **10.33 – 10.45**
  - All logging information available in *output/$NODE/stage2/003* (g4ds logging, ioids logging, database status information and console output from manual event insertion)

Listing B.4: New rule for IOIDS Data Engine Policy on node M001

```
<rule>
        <id>00150</id>
        <situation>
                <origin>remote</origin>
                <sender>M002</sender>
        </situation>

        <reactions>
                <reaction number="1">
                        <type>NewLocalEvent</type>
                        <parameters>
                                <classification>Auto</classification>
                                <community>Auto</community>
                                <distribute>
                                        <domain type="member">M003</domain>
                                </distribute>
                        </parameters>
                </reaction>
                <reaction number="99">
                        <type>Terminate</type>
                </reaction>
        </reactions>
</rule>
```

Stage 2 - Step 04 - Content and levels of details

The sources events in the SoapSy database may contain different levels of detail; meaning e.g. for a snort event information about the sensor. It was checked, whether these different levels are mirrored appropriately on the receiver‘s side of such an event as well.

08/06/2006 - Different levels of details for events

- Before this step could be executed, different sample events were created, to be inserted manually into a node‘s SoapSy database. They are made available in the output folder of this step on node M004.
- Insert events with different numbers of related information
  - After preparation, the following steps were gone through:
    - Protocol the IOIDS database status (latest event ids) on every node
    - Start the G4DS service on every node
    - Start IOIDS on every node
    - Insert the manually created events into the SoapSy database on node M004. Leave a 2 minute time gap between the two of them.
    - Stop IOIDS on every node
    - Stop the G4DS service on every node
    - Relocate available logging information into corresponding output folder
  - Time frame for this partial step: **13.23 – 13.25**
- Insert events with certain timestamp values

  - After preparation, the following steps were gone through:
    * Protocol the IOIDS database status (latest event ids) on every node
    * Start the G4DS service on every node
    * Start IOIDS on every node
    * Insert the manually created events into the SoapSy database on node M004. Leave a 2 minute time gap between them. As there were 2 events with 2 different timestamps, insert the first one first, then the second one and finally the first one again
    * Stop IOIDS on every node
    * Stop the G4DS service on every node
    * Relocate available logging information into corresponding output folder
  - Time frame for this partial step: **16.16 – 16.21**
- All logging information available in *output/$NODE/stage2/004/{a—b}* (g4ds logging, ioids logging, database status information, console output from manual event insertion and xml files for new events to be inserted)

Stage 2 - Step 05 - Performance and overhead

The last step of stage two of the experiments deals with some time and size measurement for the combination of IOIDS and G4DS in action.

09/06/2006 - Performance and Overhead

- Measure time for an event travelling through the network
  - Before the test could be run for this step, certain adjustments had to be done on data engine policies in order to set them up for passing on messages:
    * M001: pass remote messages from member M002 on to member M003
    * M003: pass remote messages from member M001 or M002 on to member M005
  - Afterwards, the step was carried out by going through the following list of actions:
    * Protocol the IOIDS database status (latest event ids) on every node
    * Start the G4DS service on every node
    * Start IOIDS on every node
    * Generate one event manually on each node from the list M004, M002, M002 and examine timestamp information from logfiles. Leave 2 minutes gap between each event.
    * Stop IOIDS on every node
    * Stop the G4DS service on every node
    * Relocate available logging information into corresponding output folder
  - Time frame for this partial step: **08.17 – 08.28**

- Measure maximal throughput of events
  - This partial step was carried out by running through the following steps:
    * Protocol the IOIDS database status (latest event ids) on every node
    * Start the G4DS service on every node
    * Start IOIDS on every node
    * Use the shell script shown in listing B.5 for generating events for the local SoapSy database with a certain time gap between each insert and keep it running for 2 minutes (hold CTRL-C for stopping). Perform five seperate runs - time gaps are 10, 5, 2, 1 and 0 seconds.
    * Stop IOIDS on every node
    * Stop the G4DS service on every node
    * Relocate available logging information into corresponding output folder
  - Time frame for this partial step: **08.52 – 09.05 & 09.33 – 10.15** (there was a break between the 5 and 2 seconds gap)
- Measure protocol overhead in the different layers of the IOIDS / G4DS system
  - For the last part of this step the network sniffer ethereal had to be deployed on nodes M001 and M004 (the command line based capturing tool tethereal). Furthermore, the IOIDS data engine on node M002 was configured to additionally send off local events to node M003 (which generates a routed event through M001).
  - Afterwards, this partial step was carried out by running through the following steps:
    * Protocol the IOIDS database status (latest event ids) on every node
    * Start the G4DS service on every node
    * Start IOIDS on every node
    * Start ethereal network sniffer on nodes M001 and M004 as shown in listing B.6
    * Insert an event into the SoapSy database on node M004 firstly and node M002 two minutes after.
    * Stop ethereal network sniffer on nodes M001 and M004 and evaluate size information from its output. Furthermore, use size information provided in G4DS and IOIDS log files.
    * Stop IOIDS on every node
    * Stop the G4DS service on every node
    * Relocate available logging information into corresponding output folder
  - Time frame for this partial step: **11.23 – 11.32**
- All logging information available in *output/$NODE/stage2/005/{a-c}* (g4ds logging, ioids logging, database status information and console output from manual event insertion)

Listing B.5: Shell script for generating load on local SoapSy database

```
while test 1:
do
python ../../soapClient.py -a localhost -i ../../insert4.xml
sleep $timegap
done
```

Listing B.6: Start ethereal for capturing G4DS traffic

```
tethereal -f ''(port 8080 or port 2000) and not broadcast and not multicast''
    -w out.dump
```

B.1.3. Stage 3

The last stage of the experiments deals with the evaluation of the overall system and the comparison of results achieved with IOIDS against other state-of-the-art approaches within the DIDS context.

After the complete details about the modifications made to the laboratory environment to suit requirement for stage three experiments, each of the following five steps within this stage will be covered in its own subsection:

- Distribution of correct event information
- Generation of high load on certain nodes
- Availability issues and Denial-of-Service resistance
- Data Engine and Access Control
- Performance and Benchmarking

Preparation

The IOIDS environment had been installed in the previous stage almost completely already. However, as this stage shall evaluate IOIDS features in comparison to other similar approaches a list of extensions and modifications had to be deployed, which are in detail:

In common for all approaches deployed in the laboratory environment are the following facts:

- High-Level installations are deployed; meaning that every node represents its own organisation (usually called managers within the approaches)
- Nodes M002, M004 and M005 act as data sources for the installations; meaning the sensors are to be installed on them
- All nodes (apart from M006) act as data sinks

- Nodes M001 and M003 are in charge to pass on messages from one domain to another (depending on the approach called gateway or relayers)
- Node M006 will be used as a penetrator for this stage only

12/06/2006 - Modifications for laboratory environment

- For the IOIDS architecture, which was in place from stages one and two already, the following modifications were applied:
  - On the sensor nodes (M002, M004 and M005) a Snort installation was installed with output plugin PGSQL enabled (version 2.3.2-3 for M004 and M005 and 2.4.3-r1 for M002). Furthermore, a database had to be created on the locally present database management systems (PostgreSQL) (Details: username is usnort; password is pwsnort and database name is snort) and the relations were created using the SQL scripts coming with the Snort distribution. The Snort IDS was configured to output event data into the prepared database on log-level. (The corresponding line within the Snort configuration file */etc/snort/snort.conf* is shown for node M001 in listing B.7)
  - The SnortDB2Soapsy data migration tool had to be installed on those machines (M002, M004, M005) in order to migrate data from the Snort database into the SoapSy database. It is provided as a resource for this project; only needs to be started - database parameters have to be configured in the configuration module *config.py* and it can be started right afterwards by calling the Python module *snortdb_to_soapsy.py*.
- In order to penetrate the network in later steps the penetration tool *Nessus* were deployed. Nessus comes as a client-server architecture; the server, which is executing the penetrations, was installed on the node M006, the client, which is coming as a graphical user interface, was deployed on machine M001. Connection between them was established through a SSH tunnel with reverse port forwarding on TCP port 1241.
- For the deployement of components of the open-source Intrusion Detection System framework Prelude the following steps were carried out:
  - On the sensor nodes M002, M004 and M005 another Snort installation was performed, this time with the output plugin *prelude* enabled. (see listing B.7 for the corresponding line within the Snort configuration file)
  - On each node (M001-M005) the following components of the Prelude framework had to be installed:
    - libprelude (0.9.9) - The essential library for Prelude; a requirement to install any other component. (requires installation for libcrypt, libgpg and gnutls)
    - libpreludedb (0.9.8) - Enables the prelude manager to log data into a database management system. (requires postgres headers to be installed) A database was created on each of the available PostgreSQL DMBS for

nodes M001-M005 (parameters uprelude, pwprelude and prelude) and the SQL script for creating the relations in there, which were provided with the prelude distribution, was executed against the database.

  * prelude-manager (0.9.4.1) - The heart of each Prelude installation. After installation it had to be configured for logging into the just created database.
  * As there was a version problem reported by the prelude-manager when trying to connect against the database, the value for the relations within the database had to be adjusted manually (from version 14.4 to 14.5).

- The Snort clients have to authenticate at the local prelude-manager whenever they shall report alerts into it; consequently, keys had to be created and the access granted to these sensors on the managers. (Check Prelude handbook ThePreludeTeam (2006) for details)
- Additionally, the Prelude management console *Prewikka* was installed on node M001 in order to access and present results more easily. For this step, the library Cheetah was installed in version 2.0rc6, yet another database had to be created (parameters: uprewikka, pwprewikka, prewikka) on node M001, which was populated with relations using the SQL file provided with Prewikka and the management console Prewikka was installed in version 0.9.5. In the end, the installed components for the Prelude environment with their versions are shown in table B.2.

Listing B.7: Enable Snort to log into certain output facilities

```
# enable alerting into PostgreSQL database
output database: log, postgresql, user=usnort password=pwsnort dbname=snort
    host=j4-itrl-12.comp.glam.ac.uk

# OR enable alerting into Prelude framework
output alert_prelude: profile=snort
```

Table B.2.: Prelude components deployed for Experiment

	M001	**M002**	**M003**	**M004**	**M005**	**M006**
Libprelude	0.9.9	0.9.9	0.9.9	0.9.9	0.9.9	
Libpreludedb	0.9.8	0.9.8	0.9.8	0.9.8	0.9.8	
Prelude-manager	0.9.4.1	0.9.4.1	0.9.4.1	0.9.4.1	0.9.4.1	
Snort		2.4.5		2.4.5	2.4.5	
Prewikka	0.9.5					

13/06/2006 - Modifications for laboratory environment continued

- To finish of the deployment of the Prelude environment relaying had to be set up between certain node in order to support inter-domain communication. Therefore, the following relays were set up:
  - Nodes M004 and M005 relay all traffic to node M003
  - Node M003 relays all traffic to M001
  - Node M002 relays all traffic to M001

  For each of those relationships, the following actions had to be gone through:
  - The receiving prelude-manager has to be set up to listen on the real ethernet interface for incoming traffic. (default is the localhost interface on 127.0.0.1)
  - The sending prelude-manager had to create a key and register with the receiving prelude-manager, which has in turn to accept this node as a data source
  - The relaying had to be enabled on the sending prelude-manager by specifying the required parameters within the prelude manager configuration file.

  Again I refer to the documentation in the Prelude Handbook (ThePreludeTeam (2006)) for detailed information on setting up the environment.
- The deployment of the SnortNet environment environment turned out to be far more difficult than expecting when defining the experiments. This started with problems in finding the corresponding implementation itself and did not stop when trying to set up snort to use the SnortNet output plugin. In the end, I had to give up on deploying a SnortNet infrastructure. The problems I was facing are discussed in more detail in the analysis part for stage 3 experiments in section 7.3.3.
- The deployment of an AirCert environment tends to be rather difficult due to the complexity of the approach. Consequently, I started up with installing some of the components on a single node:
  - libair (version 0.5.21) - the essential library for all other components; compile with postgresql support
  - pathogen (0.2.13)
  - ACID (0.9.6b23) - the analysis console for AirCert; deployed using an apache2 web-server with PHP4 and some further extensions
  - Snort with XML output plug-in; Version 2.0.0 had to be chosen as later versions do not allow XML output anymore

14/06/2006 - 17/04/2006 - Modifications for laboratory environment continued

- After these initial tests with some of the components, the AirCERT documentation "AirCERT: The Definitive Guide" (Trammell et al. (2005)) had been utilised to deploy the AirCERT infrastructure. I attempted to install all necessary components for one side on node M001, with the following steps and outcomes:
  - The libarir AirCERT library (0.5.21) - as in the run before

- The rex (0.3.13) the text analyser together with dup, which is in charge to eliminate duplicate entries
- A dredge installation (0.5.11) - the message transmitter
- An openssl installation and the configuration to establish a certificate authority on the node. Several keys and certificates had to be created and signed as indicated in the AirCERT documentations.
- An Apache installation, which is supposed to act as a receiver when started with its modules for DAV, DAVFS, SSL and URL Rewriting. Many problems had to be resolved when trying to get the constellation of these modules to work.
- Several attempts to get components working by manual invocation of them or by use of the management scripts coming with the AirCERT distribution
- Several attempts to create relations within the aircert database by using SQL scripts from any version and module available for AirCERT
- The installation of Pathogen (0.2.13) - the database deserialiser, which after sucessful compiling could not be started due to an internal error called *PGN_E_CFGPARSE*. Many attempts were made to get around the error; however, the best possible call on the command line I could think of is shown in listing B.8 has not gone through either.

In the end I decided to give up on further attempts on experiments with AirCERT due to all the problems I faced. More information and a conclusion for this decision is provided in section 7.3.1 within the analysis part.

Listing B.8: Command to start Pathogen (AirCERT module)

```
/usr/local/bin/pathogen -p /usr/local/etc/snml.pil -i ''/usr/local/apache2/
    htdocs/dav/m001-dredge/snort-snml/rex-*.xml'' -out /var/aircert/pathogen/
    out --logfile /var/log/pathogen.og --loglevel debug db=''postgresql:/
    aircert:pwaircert@193.63.129.184:5432/aircert'' sid=''1''
```

Stage 3 - Step 01 - Correct event information

The first step of the last stage deals with the processing and distribution of event information from a sensor located on a machine connected to the internet; consequently an average amount of ordinary event data should be generated.

18/06/2006 - Real world event generation

- Before this step was started, the following configurations were applied:
  - For IOIDS the data engine on M002 was configured to distribute event information into community C001
  - For Prelude IDS the relaying was configured on node M002, that node M001 is the very only parent manager for that node.

- After setting up the environment, the test was executed by going through the following steps three times, as it was essential to run this test several times due to changes occuring in the attacks pattern from the internet:
  - Capture status of IOIDS and Prelude database
  - Start G4DS and IOIDS on all nodes
  - Start the SnortDB-To-Soapsy Converter first and the Snort IDS (configured for PostgreSQL output) afterwards on the capturing node M002
  - After the pre-defined time range of 2 minutes shut down the Snort IDS first and the SnortDB-To-SoapSy converter afterwards on the capturing node M002 as soon as it was made sure that no new events were coming through for it anymore
  - After making sure that no events are processed by IOIDS anymore, shutdown IOIDS and G4DS on all nodes
  - Start the Prelude-Manager on all nodes
  - Start the Prelude-configured version of Snort on the capturing node M002
  - After the pre-defined time range of 2 minutes shutdown the Snort IDS on the capturing node M002
  - Shutdown the Prelude-Manager on all nodes
  - Store and relocate all available logging and output information in the appropriate output folder.
- Time frame for this step: *see table B.3*
- All logging information available in *output/$NODE/stage3/001/{A-C}* (g4ds logging, ioids logging and database status information)

Stage 3 - Step 02 - High load

For the second step of this stage a predefined amount of event data was generated in an isolated network segment in order to measure processing capabilites of the approaches when facing a high load of event data.

18/06/2006 - Purposeful generation of high load

- The intial setup for this step of the experiment was made up by the following actions:
  - The IOIDS configuration is made on nodes M004 and M005, which are both configured to pass on local events to all members of community C003. (M003 itself is configured to act as a data sink only in this step)
  - The Prelude setup changes comprise of the specification of M003 as relay-manager on both nodes M004 and M005 (keys had been agreed upon in the setup phase of this stage already). Node M003 is configured to not pass on any information.
- The execution of the test was carried out the following way:

  - Capture status of IOIDS and Prelude database
  - Start G4DS and IOIDS on all nodes
  - Start the SnortDB-To-Soapsy Converter first and the Snort IDS (configured for PostgreSQL output) afterwards on the capturing nodes M004 and M005
  - Execute three complete nessus scans from node M006 on nodes M004 and M005 in a row without any time break between them. The test included a full port scan and all available plugins within Nessus for finding vulnerabilities. (Each of those tests lasts about 5 minutes). Within the appendencies, figures B.1, B.2 and B.3 show screenshots of the 1) configuration, 2) excution and 3) results dialog of the nessus application.
  - Shut down the Snort IDS first and the SnortDB-To-SoapSy converter afterwards on the capturing nodes M004 and M005 as soon as it was made sure that no new events were coming through for it anymore
  - After making sure that no events are processed by IOIDS anymore, shutdown IOIDS and G4DS on all nodes
  - Start the Prelude-Manager on all nodes
  - Start the Prelude-configured version of Snort on the capturing nodes M004 and M005
  - Execute three complete nessus scans from node M006 on nodes M004 and M005 in a row without any time break between them. The test included a full port scan and all available plugins within Nessus for finding vulnerabilities. (Each of those tests lasts about 5 minutes)
  - Shutdown the Snort IDS on the capturing nodes M004 and M005
  - Shutdown the Prelude-Manager on all nodes
  - Store and relocate all available logging and output information in the appropriate output folder.
- *Note:* The IOIDS system turned out to be not capable of processing the vast amount of data. Consequently, the following components were shut down at the following situations:
  - The SnortDB-To-SoapSy Converter was shutdown after migrating 700 (M004) or 800 (M005) events from the SnortDB into the SoapSy database
  - The IOIDS system was shutdown after one hour, when it still hadn't finished to process the entire list of incoming events.
- Time frame for this step: *see table B.3*
- All logging information available in *output/$NODE/stage3/002* (g4ds logging, ioids logging and database status information)

Stage 3 - Step 03 - Availablitity and Denial-of-Service

The third step of the stage 3 experiments examined to what extend the two approaches in question were able to maintain processing when certain components of the overall topology are taken down.

Table B.3.: Start and Stop times for Experiment Execution 3-001/002

	IOIDS		**Prelude**	
	Start	*Stop*	*Start*	*Stop*
Stage 3 - 001 A	14:30	14:41	14:42	14:50
Stage 3 - 001 B	15:04	15:16	15:16	15:24
Stage 3 - 001 C	15:29	15:39	15:40	15:50
Stage 3 - 002	16:25	17:20	17:20	17:42

19/06/2006 - DOS and Availability

- The only preparation necessary was to change the settings for the Prelude-Manager on node M004 to listen on the ethernet interface instead of the localhost interface, which is enable by default.
- The execution has been devided into two parts, with the first part a node has been partially disabled, which acts as data source within the network; the second part in contrast takes partially down a data sink within the network. For each of the two parts, 4 different scenarios were carried out one after the other, namely:
  1. All components on the victim are fully functional
  2. The IDS component is taken down on the victim (IOIDS application for IOIDS approach, Prelude-Manager for Prelude appoach)
  3. The communication platform is taken down on the victim (G4DS for IOIDS approach, Prelude-Manager again for Prelude approach as it does not seperate between the two components)
  4. The network connectivity itself is slowed down by sending loads of network traffic on the TCP ports used by the approach in question.

  Each of the two parts was then carried out by going through the following steps:

  – Capture status of IOIDS and Prelude database
  – Start G4DS and IOIDS on all nodes
  – Start the SnortDB-To-Soapsy Converter first and the Snort IDS (configured for PostgreSQL output) afterwards on the capturing node M004
  – Take down the corresponding component by simply switching it of for IOIDS, G4DS or Prelude-Manager or by using the shell script as provided in listing B.9 for slowing down the network interface.
  – Run a port scan only using the Nessus application in order to generate a reasonable amount of event data.
  – After a time frame of about half a minute bring back up the taken down component.

  - Shut down the Snort IDS first and the SnortDB-To-SoapSy converter afterwards on the capturing nodes M004 and M005 as soon as it was made sure that no new events were coming through for it anymore
  - After making sure that no events are processed by IOIDS anymore, shutdown IOIDS and G4DS on all nodes
  - Start the Prelude-Manager on all nodes
  - Start the Prelude-configured version of Snort on the capturing nodes M004 and M005
  - Execute three complete nessus scans from node M006 on nodes M004 and M005 in a row without any time break between them. The test included a full port scan and all available plugins within Nessus for finding vulnerabilities. (Each of those tests lasts about 5 minutes)
  - Shutdown the Snort IDS on the capturing nodes M004 and M005
  - Shutdown the Prelude-Manager on all nodes
  - Store and relocate all available logging and output information in the appropriate output folder.
- Time frame for this step: *see table B.4*
- All logging information available in *output/$NODE/stage3/003{A-H}* (g4ds logging, ioids logging and database status information)

Listing B.9: ShellScript for generating network traffic on certain TCP ports

```
while test 1
do
  echo ``****'' | netcat grid01 4690 &> /dev/null      # for Prelude OR

  echo ``****'' | netcat grid01 2000 &> /dev/null      # together with
  echo ``****'' | netcat grid01 8080 &> /dev/null      # for IOIDS / G4DS

  sleep 0.1
  killall netcat
done
```

Stage 3 - Step 04 - Data Engine and Access Control

Step number 4 of the third stage of the practical experiments tests the approaches in the view of their mechanisms of protecting knowledge against unwanted access. Furthermore, ways are examined to distribute knowledge to a certain domain depending on certain parameters of the event.

19/06/2006 - Process and filter messages

- This step is devided into 6 parts, each of them requiring different settings to be made for the approaches on node M001; these cases with their changes are:

Table B.4.: Start and Stop times for Experiment Execution 3-003

	IOIDS		**Prelude**	
	Start	*Stop*	*Start*	*Stop*
Stage 3 - 003 A	09:10	09:27	09:27	09:29
Stage 3 - 003 B	09:40	09:48	09:49	09:55
Stage 3 - 003 C	09:59	10:06	10:06	10:09
Stage 3 - 003 D	10:30	10:36	10:36	10:39
Stage 3 - 003 E	11:12	11:17	11:17	11:20
Stage 3 - 003 F	11:25	11:30	11:30	11:33
Stage 3 - 003 G	11:40	11:46	11:47	11:50
Stage 3 - 003 H	11:54	12:01	12:02	12:04

1. In the first case all incoming events from community C002 shall be processed and integreated; information from community C001, however, shall be dropped. IOIDS data engine policies have to be configured with one rule for dropping incoming c001 traffic and passing c002 traffic. For the Prelude Manager the key for M002 has to be deleted from the list of trusted keys.
2. All incoming local events of type *generic* shall be passed on to community C002; all events of type *snort* shall be delivered to community C001 instead. For IOIDS the corresponding rules have to be integrated into the data engine policies. Prelude does not support seperation of events based on their nature.
3. All remote event information received from community C002 shall be further passed on to community C001. The IOIDS data engine policies will be modified and one rule for remote events is integrated. For prelude the manager on M003 has to be configured to pass on all information to M001; M001 in turn will forward to M002. M002 has to be configured to not pass on any messages.
4. Same as before, but also pass on messages in the opposite direction from C001 to C002. The IOIDS data engine policies on M001 get an additional rule on top of the previous one to pass on remote event from community c001 to community c002. The relay manager for Prelude on node M002 is configured with parent manager M001.
5. Messages with a low level of protection (classification) received from C002 are passed on to C001. A rule will be deployed for the IOIDS data engine (see listing B.10) for passing on this kind of messages. Prelude IDS is not capable of making distribution decisions directly based on levels of protection.
6. All messages from community C001 shall be rejected. In IOIDS configuration terms, the access control policy file will be extended by one rule, addressing this issue; for Prelude IDS the sensor ID has to be removed from the list of trusted data sources.

Furthermore, events had to be created, which could be inserted into the SoapSy

database manually using the XSM SoapClient. The following events were used:
  - The generic event from stage 2
  - An IOIDS event was saved from IOIDS before sent of and saved in a file. The timestamp information was changed to 'now', which leaves the responsibility of inserting a timestamp to the database management system.
- After setting up a single part, this one is executed by performing the following list of actions:
  - Capture status of IOIDS and Prelude database
  - Start G4DS and IOIDS on all nodes
  - Insert XML encoded events as stated in table B.5 into the SoapSy database using the SoapClient
  - After making sure that no events are processed by IOIDS anymore, shutdown IOIDS and G4DS on all nodes
  - Start the Prelude-Manager on all nodes
  - Start the Prelude-version of Snort on node M002, M004
  - Generate some event data with snort by starting up a quick port scan with the Nessus scanner on the node specified in the source prelude node in table B.5.
  - Stop the Snort IDS on node M003
  - Shutdown the Prelude-Manager on all nodes
  - Store and relocate all available logging and output information in the appropriate output folder.
- Time frame for this step: *see table B.6*
- All logging information available in *output/$NODE/stage3/004{A-F}* (g4ds logging, ioids logging and database status information)

Listing B.10: New rule for IOIDS data engine to pass on events with low protection level

```
<!-- additional rule for experiment stage 3 step 4 E -->
<rule>
        <id>00140</id>
        <situation>
                <origin>remote</origin>
                <sender>M003</sender>
                <classification>7,8,9,10</classification>
        </situation>

        <reactions>
                <reaction number="1">
                        <type>NewLocalEvent</type>
                        <parameters>
                                <classification>Auto</classification>
                                <community>Auto</community>
                                <distribute>
                                        <domain type="member">M002</domain>
                                </distribute>
                        </parameters>
                </reaction>
                <reaction number="99">
                        <type>Terminate</type>
                </reaction>
        </reactions>

</rule>
```

Table B.5.: Sources of Events for Experiment Stage 3-004

		From		
	Type	**IOIDS**	**Prelude**	**To**
A	Generic	M003	M004	M001
	Generic	M002	M002	M001
B	Generic	M001	-	M001
	Snort	M001	-	M001
C	Generic	M003	M004	M001
	Generic	M001	-	M001
	Generic	M002	M002	M001
	Snort	M003	-	M001
D	Generic	M003	M004	M001
	Generic	M002	M002	M001
E	Generic	M003	-	M001
	Snort	M003	-	M001
	Snort	M002	-	M001
F	Generic	M003	M004	M001
	Generic	M002	M002	M001
	Snort	M002	-	M001

Table B.6.: Start and Stop times for Experiment Execution 3-004

	IOIDS		**Prelude**	
	Start	*Stop*	*Start*	*Stop*
Stage 3 - 004 A	15:40	15:44	15:45	15:49
Stage 3 - 004 B	16:09	16:19		
Stage 3 - 004 C	16:39	16:48	16:48	16:52
Stage 3 - 004 D	17:31	17:36	17:37	17:42
Stage 3 - 004 E	19:13	19:21		
Stage 3 - 004 F	19:38	19:45	19:45	19:48

Stage 3 - Step 05 - Benchmarking

The very last step of the practical experiments is dealing with comparison of the two approaches regarding their performance and overhead issues.

20/06/2006 - Performance and benchmarking

- For an exact measurement it was important to be able to generate a certain amount of events within a certain time frame. The only way both approaches can accept events is through Snort; consequently, the following way was implemented to generate exactly one single event with the Snort system:
  - A new snort rule was developed, which triggers an alert for any incoming message on port 33333 (see listing B.11 for a copy of this rule)
  - Netcat had to be called to trigger exactly this rule within the snort system (the shell command is shown in listing B.12).

  Furthermore, the following changes had to be applied to the approaches in order to prepare them for this step of the experiment:
  - The IOIDS data engine on M004 had to be configured to forward locally processed event messages to M003 only.
  - The IOIDS data engine on M003 had to be configured to forward locally processed event messages to M001 only.
  - The IOIDS data engine on M001 had to be configured to forward locally processed event messages to M002 only.
  - The IOIDS data engine on M005 had to be configured to not pass on messages at all.
  - The Prelude-Manager on node M004 had to be configured to relay all events to the M003 Prelude-Manager.
  - The Prelude-Manager on node M003 had to be configured to relay all events to the M001 Prelude-Manager.

  - The Prelude-Manager on node M001 had to be configured to relay all events to the M002 Prelude-Manager.

  After executing the first part of this step (the time measurement for distribution), the following changes had to be applied again for preparing the environment for the remaining three benchmarks:

  - The IOIDS data engine on node M004 passes messages on to node M005 only.
  - The Prelude-Manager on node M004 relays messages to its only parent-manager on node M005. (keys had to be created and exchanged)

- The execution of the step was carried out by going through the following list of actions:
  - Capture status of IOIDS and Prelude database
  - Start G4DS and IOIDS on all nodes
  - Start SnortDB-To-SoapSy migration tool on M004
  - Start the version of Snort, which is logging into the PostgreSQL database on M004
  - The step of generating data is pretty much depending on the feature to measure and is explained in the list just underneath this one.
  - Stop the Snort IDS first and the SnortDB-To-SoapSy converter as soon as no data is processed with it anymore
  - After making sure that no events are processed by IOIDS anymore, shutdown IOIDS and G4DS on all nodes
  - Start the Prelude-Manager on all nodes
  - Start the Prelude-version of Snort on node M002, M004
  - The step of generating data is pretty much depending on the feature to measure and is explained in the list just underneath this one.
  - Stop the Snort IDS on node M003
  - Shutdown the Prelude-Manager on all nodes
  - Store and relocate all available logging and output information in the appropriate output folder.
- Due to the different nature of the attributes to benchmark, the generation of event changes slightly from part to part of this step. The following list provides the details, how to generate the events for each of the parts:
  - For the recording of time, an event takes to travel from the sender to the final receiver, a single event was created on M006 and directed against node M004. Directly before the time had to be taken as shown in listing B.12.
  - For the measurement of throughput several runs were necessary. The alert within the snort system was triggered again and again with configurable time gaps between each two events (as shown in the second part of listing B.12). The time gaps were decreased, starting with 10 seconds, then going through 5,2 and 1 second to end up with half a second time gap.

– The tests for overhead and confidentiality could be executed in one go as all that was needed was a network sniffer, capturing the traffic. The network sniffer was started beforehand and a single event was generated as in the first part.

Listing B.11: Snort rule for triggering a single event for incoming UDP packet

```
alert udp any any <> any 33333 (msg:''BAD-TRAFFIC tcp port 33333 traffic -
    misc rule michael''; classtype:misc-activity;sid:11543; rev:1;)
```

Listing B.12: Call of netcat to trigger new Snort rule

```
# for one single event
date && echo ''x'' | nc -u victim_address 33333

# for a number of events with X seconds time gap between two of them
date
while test 1
do
  echo ''x'' | nc -u victim_address 33333
  sleep X
done
```

B.2. More G4DS resources

Table B.7.: Specifications for laboratory environment A

	J130	**J4-12**	**J4-20**
Hardware	P4 2.8 GHz 1024 MB	P3 733 MHz 256 MB	P3 733 MHz 256 MB
OS	Gentoo 2005.0 2.6.15-r5	Debian Sarge 3.1 2.4.27-2	Debian Sarge 3.1 2.4.27-2
Networking	193.63.148.149/24	193.63.129.184/24	193.63.129.193/24 192.168.1.254/24
Hostname	j130-mp	j4-itrl-12	j4-itrl-20
Python	2.4.2	2.3.5	2.3.5
Modules			
pygresql	3.6.2	3.6.1-1	3.6.1-1
pyXML	0.8.4	0.8.4-1	0.8.4-1
SSLCrypto	0.1.1	0.1.1	0.1.1
SOAPpy	0.11.3	0.11.3-1	0.11.3-1
fpconst	0.7.1	0.7.2	0.7.2
pycrypto	2.0-r1	2.0.1	2.0.1
ezPyCrypto	0.1.1	0.1.1	0.1.1
PostgreSQL	8.0.4	7.4.7-6	7.4.7-6

Listing B.13: SQL Script for Creating G4DS relations

```
-- SQL File for creating required tables in the community database
--
-- Grid for Digital Security (G4DS)
-- Michael Pilgermann
-- mpilgerm@glam.ac.uk

--
-- Sections in here:
--  1. Communities / members
--  2. Relations for communities / members
--  3. Security stuff
--  4. Communication stuff
--  5. Services
--  6. Routing
--



--
-- 1st Section
--              Tables for communities and members
--
CREATE TABLE MEMBERS
(
    ID VARCHAR(100) PRIMARY KEY,
    NAME VARCHAR(100),
```

```
    MDL VARCHAR(100000),   -- the member description in XML
    MDLVERSION VARCHAR(30),
    MDLDATE DATE
);

CREATE TABLE COMMUNITIES
(
    ID VARCHAR(100)  PRIMARY KEY,
    NAME VARCHAR(100),
    DESCRIPTION VARCHAR(500),
    TCDL VARCHAR(100000),   -- the tc description in XML
    TCDLVERSION VARCHAR(30),
    TCDLDATE DATE
);

--
-- 2nd Section
--              Tables for relations between communities and members
--
CREATE TABLE GATEWAYS
(
    MEMBER_ID VARCHAR(100) REFERENCES MEMBERS(ID),
    SOURCE_COMMUNITY_ID VARCHAR(100) REFERENCES COMMUNITIES(ID),
    DEST_COMMUNITY_ID VARCHAR(100) REFERENCES COMMUNITIES(ID)
);

CREATE TABLE COMMUNITIES_MEMBERS
(
    MEMBERID VARCHAR(100) REFERENCES MEMBERS(ID),
    COMMUNITYID VARCHAR(100) REFERENCES COMMUNITIES(ID)
);

CREATE TABLE COMMUNITIES_AUTHORITIES
(
    MEMBERID VARCHAR(100) REFERENCES MEMBERS(ID),
    COMMUNITYID VARCHAR(100) REFERENCES COMMUNITIES(ID)
);

--
-- 3rd Section
--              Tables for security stuff
--

CREATE TABLE ALGORITHMS
(
    ID VARCHAR(100) PRIMARY KEY,
    NAME VARCHAR(50)          -- NAME such as DSA, RSA, ...
);

CREATE TABLE CREDENTIALS
(
    ID VARCHAR(100) PRIMARY KEY,
    ALGORITHMID VARCHAR(50) REFERENCES ALGORITHMS(ID),
    USERNAME VARCHAR(50),   -- user name; optional depending on algorithm
    KEY VARCHAR(10000),       -- public key
    MEMBERID VARCHAR(100) REFERENCES MEMBERS(ID)
);

-- for each algorithm "I" support, i have to provide the credentials
CREATE TABLE PERSONALCREDENTIALS
(
    ID VARCHAR(100) PRIMARY KEY,
```

```
    NAME VARCHAR(50),
    ALGORITHMID VARCHAR(100) REFERENCES ALGORITHMS(ID),
    KEY_PRIVATE VARCHAR(10000),
    KEY_PUBLIC VARCHAR(10000),
    USERNAME VARCHAR(50)    -- user name; optional depending on algorithm
);

CREATE TABLE COMMUNITIES_ALGORITHMS -- which algorithms are supported by which
    communtiy
(
    COMMUNITYID VARCHAR(100) REFERENCES COMMUNITIES(ID),
    ALGORITHMID VARCHAR(100) REFERENCES ALGORITHMS(ID)
);

--
-- 4th Section
--              Tables for communication
--
CREATE TABLE PROTOCOLS
(
    ID VARCHAR(100) PRIMARY KEY,
    NAME VARCHAR(50)        -- NAME such as SOAP, HTTP, SSH
);

CREATE TABLE ENDPOINTS      -- each endpoint is defined for a certain member within a
    certain community using a certain protocol with its specific key
(
    ID VARCHAR(100) PRIMARY KEY,
    MEMBERID VARCHAR(100) REFERENCES MEMBERS(ID),
    COMMUNITYID VARCHAR(100) REFERENCES COMMUNITIES(ID),
    PROTOCOLID VARCHAR(100) REFERENCES PROTOCOLS(ID),
    ADDRESS VARCHAR(500),   -- protocol specific address (e.g. URL for SOAP or IP/PORT
        for SSH)
    CREDENTIALID VARCHAR(100) REFERENCES CREDENTIALS(ID)
);

CREATE TABLE COMMUNITIES_PROTOCOLS  -- which protocols are supported by which
    community
(
    COMMUNITYID VARCHAR(100) REFERENCES COMMUNITIES(ID),
    PROTOCOLID VARCHAR(100) REFERENCES PROTOCOLS(ID)
);

--
-- 5th Section
--              Tables for Services and their relations
--
CREATE TABLE SERVICES
(
    ID VARCHAR(100) PRIMARY KEY,
    NAME VARCHAR(50),
    KSDL VARCHAR(100000),       -- Service description in Knowledge Service
        Description language
    KSDLVERSION VARCHAR(30),
    KSDLDATE DATE       -- Date of this version of the KSD
--    WSDL VARCHAR(10000),      -- Service description in Web Service Description
    Language
--    WSDLVERSION VARCHAR(30), -- Version of the file, not the WSDL version itself!
--    WSDLDATE DATE       -- Date of this version of teh WSD
);

CREATE TABLE SERVICES_COMMUNITIES
```

```
(
    SERVICEID VARCHAR(100) REFERENCES SERVICES(ID),
    COMMUNITYID VARCHAR(100) REFERENCES COMMUNITIES(ID)
);

CREATE TABLE SERVICES_MEMBERS
(
    SERVICEID VARCHAR(100) REFERENCES SERVICES(ID),
    MEMBERID VARCHAR(100) REFERENCES MEMBERS(ID)
);

CREATE TABLE SERVICES_AUTHORITIES
(
    SERVICEID VARCHAR(100) REFERENCES SERVICES(ID),
    MEMBERID VARCHAR(100) REFERENCES MEMBERS(ID)
);

--
-- 6th Section
--              Tables for routing
--
CREATE TABLE ROUTINGTABLE
(
    ID VARCHAR(100) PRIMARY KEY,
    SOURCE_COMMUNITY VARCHAR(100) REFERENCES COMMUNITIES(ID),
    DESTINATION_COMMUNITY VARCHAR(100) REFERENCES COMMUNITIES(ID),
    GATEWAY_MEMBER_ID VARCHAR(100) REFERENCES MEMBERS(ID),
    GATEWAY_COMMUNITY_ID VARCHAR(100) REFERENCES COMMUNITIES(ID),
    COSTS INT
);
```

Table B.8.: Specifications for laboratory environment B

	Grid01	**Grid02**	**Grid03**
Hardware	P3 800 MHz 256 MB	P3 733 MHz 256 MB	P3 733 MHz 256 MB
OS	Debian Sarge 3.1 2.4.27-2	Debian Sarge 3.1 2.4.27-2	Debian Sarge 3.1 2.4.27-2
Networking	192.168.1.1/24	192.168.1.2/24	192.168.1.3/24
Hostname	grid01	grid02	grid03
Python	2.4.2	2.3.5	2.3.5
Modules			
pygresql	3.6.1-1	3.6.1-1	3.6.1-1
pyXML	0.8.4-1	0.8.4-1	0.8.4-1
SSLCrypto	0.1.1	0.1.1	0.1.1
SOAPpy	0.11.3-1	0.11.3-1	0.11.3-1
fpconst	0.7.2	0.7.2	0.7.2
pycrypto	2.0.1	2.0.1	2.0.1
ezPyCrypto	0.1.1	0.1.1	0.1.1
PostgreSQL	7.4.7-6	7.4.7-6	7.4.7-6

Listing B.14: Installation instructions for G4DS

```
INSTALLATION of G4DS

Grid for Digital Security
Michael Pilgermann
mpilgerm@glam.ac.uk

Content
-------
- Requirements
- Before installation
- Unpacking
- Distribution
- Configuration
- Start & Stop
- Uninstall

Requirements
------------
* Python Installation (http://www.python.org)
* PostGreSQL Database (http://www.postgresql.org/)
* python site package 'pygresql' for connecting against the database (http://www.
    pygresql.org/)
* python site package 'PyXML' (http://pyxml.sourceforge.net/)
* python site package 'cElementTree' (http://effbot.org/zone/celementtree.htm)
* python site package 'SSLCrypto' (http://www.freenet.org.nz/python/SSLCrypto/)
    - depending on an openssl installation (source / headers)
* for SOAP protocol implementation
    - site package 'SOAPpy' (http://pywebsvcs.sourceforge.net/)
    - site package 'fpconst' (http://research.warnes.net:9090/~warnes/fpconst/)
* for encryption algorithms
    - site package 'pycrypto' - RSA, ElGamal (http://www.amk.ca/python/code/crypto.
        html)
        There is a conflict between several versions of PyCrypto. Make sure you are
            running
        the same version on all nodes. Current Version here: pycrypto-2.0-r1
        Furthermore, make sure you install the full PyCrypto version (otherwise,
            ezPyCrypto will
        fail to load) - some distributions (such as Ubuntu) do not provide all
            algorithms by default and
        you have to download and install manually.
    - site package 'ezPyCrypto' - RSA, ElGamal (http://www.freenet.org.nz/ezPyCrypto/)
* Gentoo's python module output.py (for colored console output)
    - if you are not running gentoo, copy the file 'output.py' from the sub directory
        'g4ds/thridparty' into the main folder 'g4ds'

Before installation
-------------------
* The database has to be prepared (see also help file DB_HOWTO for more details)
    - create a user (default name 'ug4ds' / default password 'pwg4ds')
    - create a database (default name 'g4ds')
    - create the required tables in the database using the provided script 'sql/
        createtables.sql'

Unpacking
---------
* Unpack the archive to a location of your choice

Distribution
------------
The following steps have to be performed with root priveleges!
```

```
    su -    (password)

* Install G4DS as a python site package
    - run the setup module in the g4ds directory with option install:
      python setup.py install
* Check permissions on the files
    - init script "/etc/init.d/g4dsrc" should be executable for "root"
        chmod u+x /etc/init.d/g4dsrc
    - g4ds listener module "/usr/sbin/g4dslistener.py" should be executable for "root"
        chmod u+x /usr/sbin/g4dslistener.py
* Want to start G4DS backend at bootup time?
    - For most Linux distributions: Link from your directory of the default boot level
        to the g4dsrc script:
        ln -s /etc/init.d/g4dsrc /etc/rc.d/rc3.d/99g4dsrc   (assuming, 3 is your
            default run level)
    - For Gentoo Linux
        rc-update add g4dsrc default
    - For Debian Linux
        update-rc.d g4dsrc defaults

Configuration
-------------
* Edit the configuration file g4ds.conf in the systems settings folder "/etc"
    - apply your settings for your organisation at the top of the file (name,
        organisation, ...) - will be part of your member description
    - apply your settings for the database host and port
    - if you did not use default user / password / database; apply the settings here
* In the same file further down, edit the configuration file for protocols
    - check the settings for the protocols, especially the local IP address and ports
* Run the install module 'install.py' from the g4ds directory
    - This prepares your local node for G4DS by distributing the knowledge of the
        config files to the managers and database

Start & Stop
------------
* Starting G4DS
    /etc/init.d/g4dsrc start
* Stopping G4DS
    /etc/init.d/g4dsrc stop

Uninstall
---------
* Run the sql script for deleting all the tables 'sql/droptables.sql'
* Delete the g4ds directory with all subdirectories
* Delete the file "/etc/init.d/g4dsrc" (and possible links to it in your runlevel
    directories)
* Delete the g4ds listener module "g4dslistener.py" from the directory "/usr/sbin"
* Delete the site package g4ds from your local python installation
(sorry for not providing something more handy here yet)
```

Listing B.15: Output for G4DS Installation on M004

```
Installation started
   Check for availability of required modules
      output - colored console output                                          [  OK  ]
      pygresql - postgresql database connector                                 [  OK  ]
      PyXML - xml processing libraries                                         [  OK  ]
      fpconst - required module for soap                                       [  OK  ]
      SOAPpy - SOAP implementation                                             [  OK  ]
      pycrypto - Low level cryptography toolkit                                [  OK  ]
      ezPyCrypto - High level cryptography api                                 [  OK  ]
   Finished checking of modules                                                [  OK  ]
      Configuration file: /etc/g4ds.conf                                       [  OK  ]
   Initialise member database with myself as the only member
      Create new member                                                        [  OK  ]
      Add member to local manager                                              [  OK  ]
   New member (M004) finished.                                                 [  OK  ]
   Initialise community database with one initial entry
      Initialise temporary entries for back referencing
         Adding temporare member 'M001' (authority) to system and community.   [  OK  ]
         Adding temporare member 'M002' (authority) to system and community.   [  OK  ]
      Members temporaly added.                                                 [  OK  ]
      Preparing new Community 'Default Community'
         Adding member 'M001' (authority) to system and community.             [  OK  ]
         Adding member 'M002' (authority) to system and community.             [  OK  ]
      Community prepared.                                                      [  OK  ]
      Apply community description and add members                              [  OK  ]
         Add local member to the community                                     [  OK  ]
      Finshed new Community 'Default Community'                                [  OK  ]
   Finished community database                                                 [  OK  ]
   Initialise algorithms
      Algorithm 'elgamal'                                                      [  OK  ]
      Algorithm 'rsa'                                                          [  OK  ]
   Finished algorithms                                                         [  OK  ]
   Initialise credentials for algorithms
      Personal Credential for algorithm 'elgamal'                              [  OK  ]
      Public Credential for algorithm 'elgamal'                                [  OK  ]
      Personal Credential for algorithm 'rsa'                                  [  OK  ]
      Public Credential for algorithm 'rsa'                                    [  OK  ]
   Finished credentials                                                        [  OK  ]
   Initialise protocols and their endpoints
      Add Protocol 'tcpsocket'                                                 [  OK  ]
      Add Endpoints for protocol 'tcpsocket'
         Add Endpoint for protocol 'tcpsocket' and Algorithm 'elgamal'         [  OK  ]
         Add Endpoint for protocol 'tcpsocket' and Algorithm 'rsa'             [  OK  ]
      Add Protocol 'soap'                                                      [  OK  ]
      Add Endpoints for protocol 'soap'
         Add Endpoint for protocol 'soap' and Algorithm 'elgamal'              [  OK  ]
         Add Endpoint for protocol 'soap' and Algorithm 'rsa'                  [  OK  ]
   Finished protocols and endpoints                                            [  OK  ]
   Generate and store member description for the local node                     [  OK  ]
Installation finished                                                          [  OK  ]
```

Listing B.16: Community Description for C001

```
<?xml version="1.0" encoding="UTF-8"?>
<tcdl>
    <id>C001</id>
    <version>1.0.0.0</version>
    <name>g4ds_01</name>
    <creationdate>2006-05-16</creationdate>

    <description>
        <fullname>G4DS Evaluation - community 01</fullname>
        <organisation>University of Glamorgan</organisation>
        <location>
            <country>
                <code>UK</code>
                <name>United Kingdom</name>
            </country>
            <city>Cardiff</city>
        </location>
    </description>

    <communication>
        <protocols>
            <protocol>
                <name>soap</name>
                <comment>SOAP, as implemented for G4DS. Server must listen to incoming
                    messages on function 'newMessage'</comment>
            </protocol>
            <protocol>
                <name>tcpsocket</name>
                <comment>Simple communication over TCP sockets as implemented in G4DS
                    / protcols for TCP sockets</comment>
            </protocol>
        </protocols>
        <algorithms>
            <algorithm>
                <name>rsa</name>
                <comment>RSA, as implemented in G4DS / algorithms</comment>
            </algorithm>
            <algorithm>
                <name>elgamal</name>
                <comment>ElGamal, as implemented in G4DS / algorithms</comment>
            </algorithm>
        </algorithms>
    </communication>

    <authorities>
        <authority>
            <memberid>M001</memberid>
            <endpoint>
                <protocol>soap</protocol>
                <address>http://193.63.129.184:8080</address>
                <credential>
                    <algorithm>rsa</algorithm>
                    <publickey>
                        <![CDATA[3c [...] 0a]]>
                    </publickey>
                </credential>
            </endpoint>
        </authority>
        <authority>
            <memberid>M002</memberid>
```

```
            <endpoint>
                <protocol>soap</protocol>
                <address>http://193.63.148.149:8080</address>
                <credential>
                    <algorithm>rsa</algorithm>
                    <publickey>
                        <![CDATA[3c [...] 0a]]>
                    </publickey>
                </credential>
            </endpoint>
        </authority>
    </authorities>

    <routing>
        <gateways>
            <incoming>
                <gateway>
                    <memberid>M001</memberid>
                    <source>
                        <communityid>C002</communityid>
                    </source>
                </gateway>
            </incoming>
            <outgoing>
                <gateway>
                    <memberid>M001</memberid>
                    <destination>
                        <communityid>C002</communityid>
                    </destination>
                </gateway>
            </outgoing>
        </gateways>
    </routing>

</tcdl>
```

Listing B.17: Python program for chat test service

```
"""
Simple Test Service for G4DS

Grid for Digital Security (G4DS)

All it does is waiting for user input and sending it to the requested destination
    member.

@author: Michael Pilgermann
@contact: mailto:mpilgerm@glam.ac.uk
@license: GPL (General Public License)
"""

import g4ds.g4dsservice
from g4ds.errorhandling import G4dsException

class TestService:
    """
    Class, which holds the callback for incoming messages and the functions user input
         for processing
    messages to be passed on to the g4ds system.
    """

    def __init__(self, serviceid = 'S123456'):
        """
        Connects against G4ds backend.
        """
        self._gs = g4ds.g4dsservice.G4dsService()
        try:
            self._gs.connect(serviceid, None, 'key', callback = self.callback)
            self.userInput()
        except G4dsException, msg:
            print "Could not connect against G4DS: %s" %msg

    def callback(self, msg, metadata):
        """
        Callback for incoming messages  from g4ds.

        Just prints the msg to std out.
        """
        from g4ds.g4dsservice import METADATA_SENDERID
        import time
        print "\n\tReceived (%s @%s): %s" %(metadata[METADATA_SENDERID], time.strftime
            ('%x-%X'), msg)
##          print "\tInfo: %s" %(metadata)


    def userInput(self):
        """
        Request user input and send it over.
        """

        while 1:
            msg = raw_input('Message (q to quit): ')
            if msg == 'q':
                self._gs.disconnect()
                break
            receiver = raw_input('Member ID of receiver: ')
            try:
                self._gs.sendMessage(receiver, None, msg, actionstring='chat.send.
```

```
                    message')
            except G4dsException, msg:
                print "\nSomething went wrong here - error message from G4DS: %s"


if __name__ == "__main__":
    import sys      # this way you can specify a service id on the prompt - testing
        several services
    if len(sys.argv) > 1:
        ts = TestService(sys.argv[1])
    else:
        ts = TestService()
```

Listing B.18: Service Description for Test-Chat service

```
<?xml version='1.0' encoding='UTF-8'?>
<ksdl>
    <id>S123456</id>
    <version>1.0.0.0</version>
    <name>Simple g4ds test - chatting service</name>
    <creationdate>2006-05-16</creationdate>
    <lastupdate>2006-05-16</lastupdate>

    <description>
        <fullname>G4DS test service - simple chat program</fullname>
        <contacts>
            <contact>
                <name>Michael Pilgermann</name>
                <organisation>University of Glamorgan / Information Secury Research
                    Group</organisation>
                <email>mpilgerm@glam.ac.uk</email>
            </contact>
        </contacts>
    </description>

    <communication>
        <communities>
            <community>
                <id>C001</id>
            </community>
            <community>
                <id>C002</id>
            </community>
            <community>
                <id>C003</id>
            </community>
        </communities>

        <messageformats>
            <messageformat>
                <id>S123456fulltext</id>
                <name>Chatting full text messages</name>
                <definition/>
            </messageformat>
        </messageformats>
    </communication>

    <authorities>
        <authority>
            <memberid>M001</memberid>
        </authority>
        <authority>
            <memberid>M002</memberid>
        </authority>
    </authorities>
</ksdl>
```

Listing B.19: G4DS logging outupt on node M001 for experiment stage 1 step 001b

```
2006-05-18 10:59:02 000 G4DS Logging started (level 5)
2006-05-18 10:59:04 799 Routing Table updater intialised
2006-05-18 10:59:07 998 Client connected for service S123456.
2006-05-18 10:59:58 199 New incoming message
2006-05-18 10:59:59 198 -- MSG ID Z261767 | SENDER M002
2006-05-18 10:59:59 198 -- Size of msg (brutto | netto): 6260 | 525 Bytes
2006-05-18 10:59:59 198 -- Service Msg - Service test service (S123456)
2006-05-18 10:59:59 698 Access Control - message passed: M002 -> S123456 (A: g4ds.
    service.S123456)
2006-05-18 10:59:59 698 Access Control - message passed: M002 -> S123456 (A: chat.send
    .message)
2006-05-18 11:00:00 198 -- Service Msg - passed message to connected client.
2006-05-18 11:00:12 199 New incoming message
2006-05-18 11:00:13 198 -- MSG ID Z293157 | SENDER M002
2006-05-18 11:00:13 198 -- Size of msg (brutto | netto): 26556 | 7162 Bytes
2006-05-18 11:00:13 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:00:13 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:00:13 298 Sending control message
2006-05-18 11:00:14 299 New outgoing message - direct delivery (M003 | C002)
2006-05-18 11:00:14 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:00:14 296 -- Size of Data 26716 chars
2006-05-18 11:00:30 199 New incoming message
2006-05-18 11:00:30 198 -- MSG ID Z680649 | SENDER M002
2006-05-18 11:00:30 198 -- Size of msg (brutto | netto): 32710 | 9090 Bytes
2006-05-18 11:00:30 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:00:30 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:00:30 298 Sending control message
2006-05-18 11:00:31 299 New outgoing message - direct delivery (M003 | C002)
2006-05-18 11:00:31 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:00:31 296 -- Size of Data 32894 chars
2006-05-18 11:00:48 199 New incoming message
2006-05-18 11:00:49 198 -- MSG ID Z588422 | SENDER M002
2006-05-18 11:00:49 198 -- Size of msg (brutto | netto): 6272 | 533 Bytes
2006-05-18 11:00:49 198 -- Service Msg - Service test service (S123456)
2006-05-18 11:00:49 698 Access Control - message passed: M002 -> S123456 (A: g4ds.
    service.S123456)
2006-05-18 11:00:49 698 Access Control - message passed: M002 -> S123456 (A: chat.send
    .message)
2006-05-18 11:00:51 198 -- Service Msg - passed message to connected client.
2006-05-18 11:01:16 199 New incoming message
2006-05-18 11:01:16 199 New incoming message
2006-05-18 11:01:17 198 -- MSG ID Z757657 | SENDER M002
2006-05-18 11:01:17 198 -- Size of msg (brutto | netto): 26660 | 7196 Bytes
2006-05-18 11:01:17 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:01:17 198 -- MSG ID Z19840 | SENDER M002
2006-05-18 11:01:17 198 -- Size of msg (brutto | netto): 6282 | 527 Bytes
2006-05-18 11:01:17 198 -- Service Msg - Service test service (S123456)
2006-05-18 11:01:17 698 Access Control - message passed: M002 -> S123456 (A: g4ds.
    service.S123456)
2006-05-18 11:01:17 698 Access Control - message passed: M002 -> S123456 (A: chat.send
    .message)
2006-05-18 11:01:18 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:01:18 298 Sending control message
2006-05-18 11:01:18 198 -- Service Msg - passed message to connected client.
2006-05-18 11:01:18 299 New outgoing message - direct delivery (M003 | C002)
```

```
2006-05-18 11:01:18 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:01:18 296 -- Size of Data 26822 chars
2006-05-18 11:01:49 199 New incoming message
2006-05-18 11:01:50 199 New incoming message
2006-05-18 11:01:50 199 New incoming message
2006-05-18 11:01:50 199 New incoming message
2006-05-18 11:01:51 198 -- MSG ID Z778370 | SENDER M002
2006-05-18 11:01:51 198 -- Size of msg (brutto | netto): 26822 | 7214 Bytes
2006-05-18 11:01:51 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:01:51 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:01:51 298 Sending control message
2006-05-18 11:01:52 198 -- MSG ID Z128199 | SENDER M002
2006-05-18 11:01:52 198 -- Size of msg (brutto | netto): 32638 | 9168 Bytes
2006-05-18 11:01:52 198 -- MSG ID Z787291 | SENDER M002
2006-05-18 11:01:52 198 -- Size of msg (brutto | netto): 32724 | 9138 Bytes
2006-05-18 11:01:52 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:01:52 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:01:52 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:01:52 298 Sending control message
2006-05-18 11:01:53 299 New outgoing message - direct delivery (M003 | C002)
2006-05-18 11:01:53 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:01:53 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:01:53 298 Sending control message
2006-05-18 11:01:53 296 -- Size of Data 26938 chars
2006-05-18 11:01:53 198 -- MSG ID Z848583 | SENDER M002
2006-05-18 11:01:53 198 -- Size of msg (brutto | netto): 32654 | 9136 Bytes
2006-05-18 11:01:53 198 -- Control Msg - SS: Routing Engine
2006-05-18 11:01:53 698 Access Control - message passed: M001 -> C002 (A: g4ds.routing
    .route)
2006-05-18 11:01:53 298 Sending control message
2006-05-18 11:01:53 299 New outgoing message - direct delivery (M003 | C002)
2006-05-18 11:01:53 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:01:53 296 -- Size of Data 32802 chars
2006-05-18 11:01:54 299 New outgoing message - direct delivery (M003 | C002)
2006-05-18 11:01:54 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:01:54 296 -- Size of Data 32910 chars
2006-05-18 11:01:54 299 New outgoing message - direct delivery (M003 | C002)
2006-05-18 11:01:54 296 -- Endpoint Endpoint (E866658): MemberID is M003. Address:
    http://193.63.129.193:8080.
2006-05-18 11:01:54 296 -- Size of Data 32854 chars
2006-05-18 11:02:10 997 Client disconnected for service S123456.
2006-05-18 11:02:51 799 Routing Table updater is shutdown
2006-05-18 11:02:51 000 G4DS Logging shut down
```

Listing B.20: Python program for changing member id inside message

```
# Python program to change the sender ID of a g4ds message
#
# - private key must be present in the local database (parameters hard coded)
# - id of this key has to be provided on the shell
#
# Michael Pilgermann (mpilgerm@glam.ac.uk)
# 22/05/2006
#


def getKey(keyid):
        print "Receive key from database ..."
        import pg
        connection = pg.connect('g4ds', 'j4-itrl-12', 5432, None, None, 'ug4ds', '
            pwg4ds')

        query = """select id, name, key_private, key_public from personalcredentials
            where id = '""" + keyid + """'""";
        ##print query
        result = connection.query(query)

        list = result.getresult()
        if not len(list):
                print "No key found with given ID"
                return None, None
        for item in list:
            id = item[0]
            name = item[1]
            key = item[2]
            key_pub = item[3]

        print "  Found key with name <%s>" %(name)
        return key, key_pub

def getContent(srcfilename):
        """
        Actually we should parse the XML properly here - but too much hassles; let's
            just look for the CDATA section ...
        """
        print "Open file and extract information ..."
        st = "<![CDATA["
        file = open(srcfilename, 'r')
        content = file.read()
        file.close()
        startpos = content.index(st)+9
        endpos = content.index(']]>', startpos)
        hexdata = content[startpos:endpos]
        print "  Found data. Let's convert and unpack ..."
        import binascii as hex
        zipped = hex.unhexlify(hexdata)
        import zlib
        unzipped = zlib.decompress(zipped)
        print "  OK"
        return unzipped

def decrypt(content, key):
        print "Time to decrypt ..."
        import ezPyCrypto
        key = ezPyCrypto.key(key)
        plain = key.decStringFromAscii(content)
```

```
        print "  Done"
        return plain

def replaceMemberId(content, oldId, newId):
        print "Replacing member id ..."
        pos = content.index("<senderid>" + oldId + "</senderid>")
        st = content[:pos]
        st +="<senderid>" + newId + "</senderid>"
        st += content[pos + len("<senderid>" + oldId + "</senderid>"):]
        print "  Done"
        return st

def encrypt(content, key_pub):
        print "Encrypt again with receiver's public key ..."
        import ezPyCrypto
        key = ezPyCrypto.key()
        key.importKey(key_pub)
        cipher = key.encStringToAscii(content)
        print "  Done"
        return cipher

def packandwrite(content, srcFileName, dstFileName):
        print "Time to wrap it in a g4ds message ..."
        import zlib
        zipped = zlib.compress(content)
        import binascii as hex
        hexed = hex.hexlify(zipped)

        st = "<![CDATA["
        file = open(srcFileName, 'r')
        filecontent = file.read()
        file.close()
        startpos = filecontent.index(st)+9
        endpos = filecontent.index(']]>', startpos)

        outfile = open(dstFileName, 'w')
        outfile.write(filecontent[:startpos])
        outfile.write(hexed)
        outfile.write(filecontent[endpos:])
        outfile.close()
        print "  OK"
        return


import sys
if len(sys.argv) != 6:
        print "You have to provide the ID of the private key to decrypt the message,
            the name of the file containing the message, the original and the fake
            member id and the destination filename!"
        print "\tUsage: %s $KEYID $SRC_FILENAME $REAL_MEMBERID $FAKE_MEMBERID
            $DEST_FILENAME" %(sys.argv[0])
        sys.exit(1)
key, key_pub = getKey(sys.argv[1])
if not key:
        print "No key found with the given ID. Sorry - I quit"
        sys.exit(1)
content = getContent(sys.argv[2])
plain = decrypt(content, key)

newPlain = replaceMemberId(plain, sys.argv[3], sys.argv[4])
if not newPlain:
        sys.exit(1)
```

```
cipher = encrypt(newPlain, key_pub)

packandwrite(cipher, sys.argv[2], sys.argv[5])
```

B.3. More IOIDS resources

Listing B.21: Installation instructions for IOIDS

```
INSTALLATION of IOIDS

Inter-Organisational Intrusion Detection System (IOIDS)
Michael Pilgermann
mpilgerm@glam.ac.uk

Content
-------
- Requirements
- Installation & Configuration
- Start
- Uninstall

Requirements
------------
- Postgres Database (tested on version 8)
- Python 2.3+
- G4DS
- SoapSy
- PyXML
- SoapXML RPC by Konstantinos Xynos (kxynos@glam.ac.uk) (now coming with this package)
- SoapSy Tools (download from j4-itrl-12.comp.glam.ac.uk/g4ds#download)

Installation & Configuration
----------------------------


1) G4DS preparations
- install g4ds first (from now we assume it's working)
- create a private key with g4ds, which we use later on for connecting (authenticating
    ) against it (maintain in g4ds)
    * save the private key at the location stated in your ioids config file
- make this service known to g4ds
    * run g4ds maintain and apply the service description (provided in the sub folder
        descriptions) to the g4ds system

2) Unpack archive to location of your choice
- change into this directory

3) Database preparations
- SoapXML RPC database connector is now included in the IOIDS package - it is
    installed automatically, ones you run the setup of ioids (#4)
- Create an SQL script using the provided XDS program
    * run XDS python program and create SQL script this way
        python thirdparty/soap_db/soap_server/XDS.py -i desriptions/
            IOIDS_SoapSy_DatabaseSchema.xml -o descriptions/
            IOIDS_SoapSy_DatabaseSchema.sql
- Change os user to postgres
    * su - postgres
- Create user in database mangement system
    * createuser -P uioids
    <type password 'pwioids'>
- Create database for ioids
    * createdb -O uioids ioids
    * exit (leave user postgres)
- Run the provided SQL Script and create relations in database
    * psql -U uioids [-h localhost] ioids -f descriptions/IOIDS_SoapSy_DatabaseSchema.
        sql

4) Installation
```

```
- Distribute the files over the local filesystem (sitepackages, config files and
    programs) - as root
    * su
    * python setup.py install
    * exit
- check and adjust permissions for rc-script and python script
    * chmod 755 /etc/init.d/xsmrc
    * chmod 755 /usr/bin/XSM.py
- Make the IOIDS access control policies known to the G4DS system
    * copy policy files (descriptions/*.pol) to G4DS policy system folder (default /
        var/lib/g4ds/policies)
    * register files in g4ds configuration file (default /etc/g4ds.conf - value
        POLICY_FILES)
    * apply changes to current g4ds policies for activation (see descriptions/
        ioids_g4ds_policy.pol for details)

5) Configuration
- edit the global configuration file (config.py)
    * check everything in there and adjust to your needs
- edit the SoapXML RPC database configuration file
    * check settings in /etc/XSM-configuration.xml

Start
-----
- start G4DS (as root)
    /etc/init.d/g4dsrc start
- start SoapXML RPC database (as root)
    /etc/init.d/xsmrc start
- start ioids
    python ioids.py


Uninstall
---------
coming soon
```

B.4. Other figures and listings

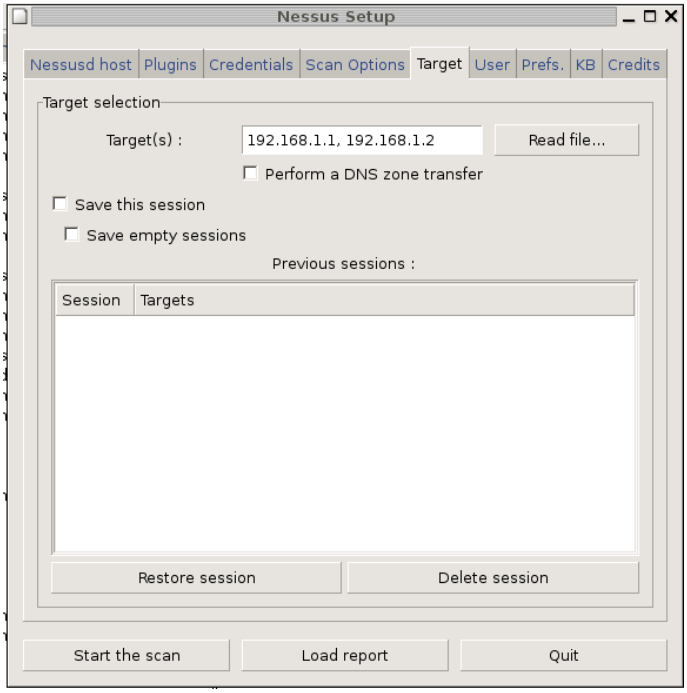


Figure B.1.: Nessus configuration

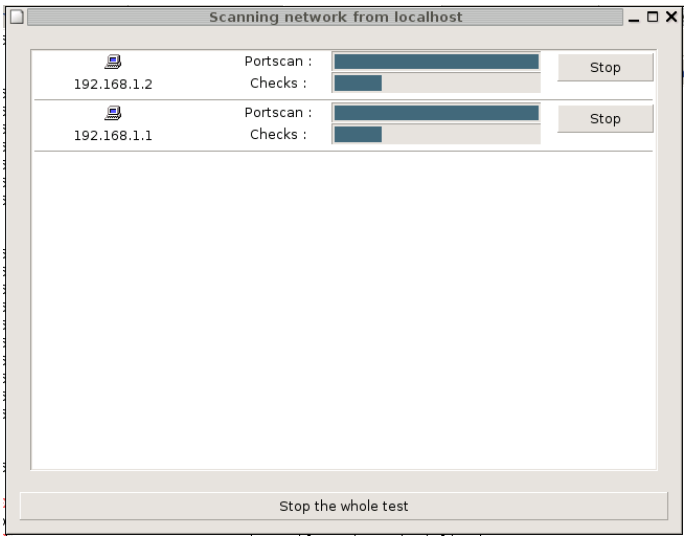


Figure B.2.: Nessus during execution

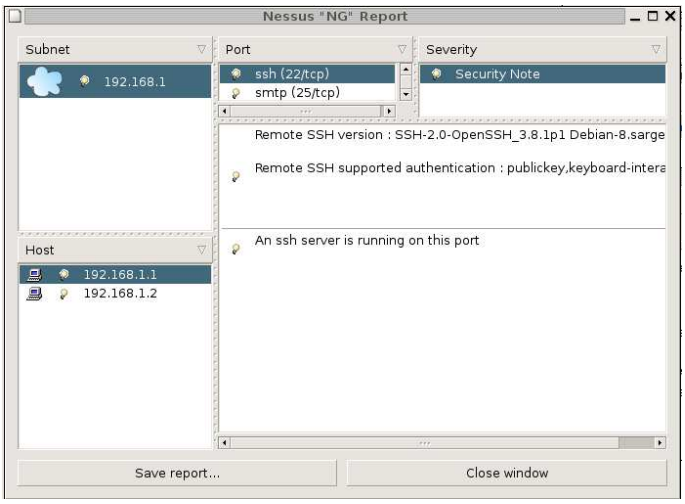


Figure B.3.: Nessus results

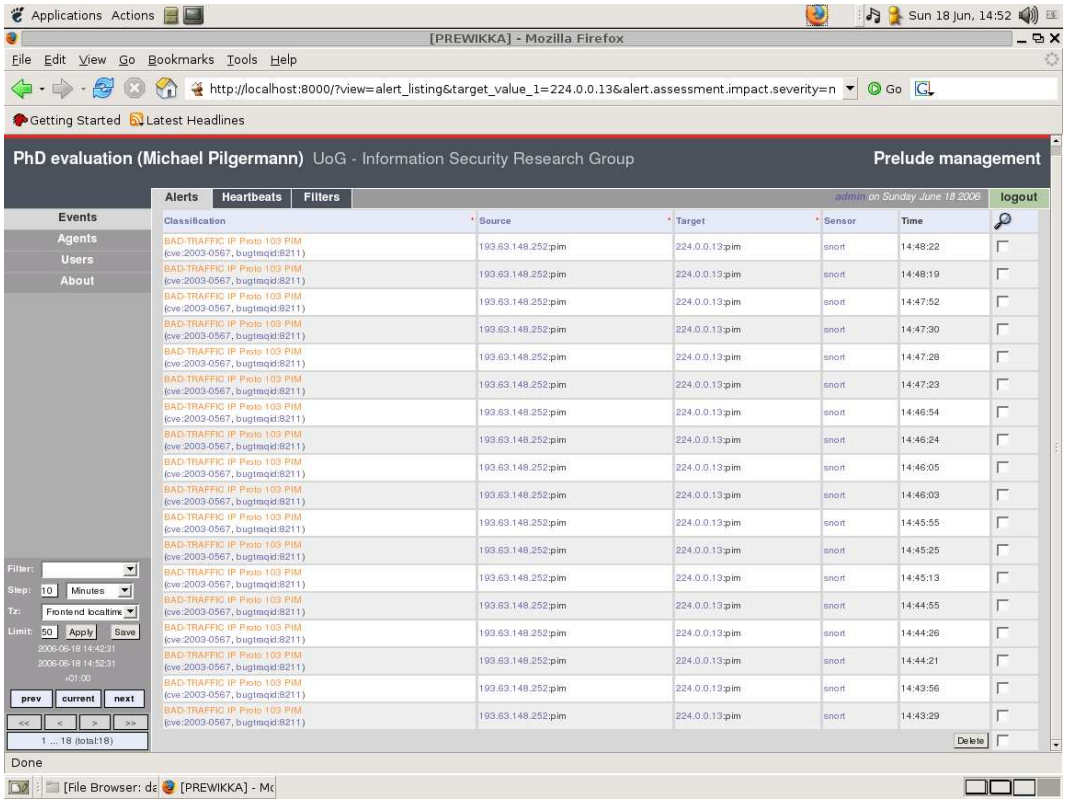


Figure B.4.: Prelude analysis console *PreWikka* detailed event information