A system for collection and analysis of forensic evidence

Norsk Regnesentral
ANVENDT DATAFORSKNING
Norwegian Computing Center/Applied Research and Development

NOTAT/NOTE

GEM/05/02

Jerker Danielsson
Knut Håkon T. Mørch
Per Røe

11.02.2003

Norsk Regnesentral
ANVENDT DATAFORSKNING

Notat/Note

Tittel/Title:
A system for collection and analysis of forensic evidence

Dato/Date: February
År/Year: 2003
Notat nr:
Note no: GEM/05/02

Forfatter/Author:
Jerker Danielsson, Knut Håkon Tolleshaug Mørch, Per Røe

Sammendrag/Abstract:
On today's Internet, attackers are seldom made accountable for their actions in a court of law. To make attackers accountable for their actions is a vital part of the struggle for defeating cyber crime. One of the reasons for the lack of legal action against attackers is that the collection and analysis of forensic evidence is a very troublesome and time-consuming activity.

The work presented in this report is based on the hypothesis that a module based distributed system where every module is specialised at collecting forensic evidence in a specific environment will simplify the collection and increase the quality of forensic evidence. The collection will be trigged, refined and coordinated based on the intrusion detection information supplied by IDS technology. In a large network it is not possible to continuously collect all the data needed for forensic analysis due to performance and the enormous amount of data. The approach we propose circumvent this problem by collecting fine-grained data only when needed and only at the hosts and network devices affected by the attack.

This report investigates this hypothesis and its implications. An analysis of which data to collect and the requirements that has to be put on a system for collecting forensic evidence is presented. Much of this analysis is generally applicable to systems for collecting digital forensic evidence and not just the system presented in this report. A design of a system implementing the hypothesis has been developed, this design and a prototype implementation of the system are also presented. The report concludes with the presentation of a number of future research directions, related to the work presented in this report.

Emneord/Keywords: IDS, Computer Forensics, Logging, Auditing

Målgruppe/Target group: NR, research institutions

Tilgjengelighet/Availability: Open

Prosjektnr./Project no.: 802022

Satsningsfelt/Research field: Computer Security

Norsk Regnesentral / Norwegian Computing Center
Gaustadalléen 23, Postboks 114 Blindern, N-0314 Oslo, Norway
Telefon (+47) 22 85 25 00, telefax (+47) 22 69 76 60

Antall sider/No. of pages: 78

Table of contents

1 Introduction ... 3
1.1 Project background ... 3
1.1.1 Collection of forensic evidence ... 3
1.1.2 Enforcement of security properties ... 4
1.2 Relevant technologies ... 5
1.2.1 Intrusion Detection Systems (IDS) ... 5
1.2.2 Computer Forensics ... 6
1.3 Hypothesis ... 7
1.3.1 Concept scenario ... 7
1.3.2 Scope ... 8
1.3.3 Research challenges ... 8
1.4 Structure of the report ... 9
2 Related work ... 10
3 IDS issues ... 12
3.1 False negatives ... 12
3.2 Collection delay ... 12
3.3 Denial of service ... 13
3.4 One attack is possibly detected as many attacks ... 13
4 Analysis of which data to collect ... 14
4.1 Analysis ... 15
4.2 Data suitable to be collected under the fine-grained hypothesis ... 15
4.3 Collection candidates ... 17
4.3.1 System calls ... 18
4.3.2 File access timestamps and file content ... 20
5 Requirements ... 22
5.1 System Functions ... 22
5.2 System Attributes ... 22
5.2.1 Security requirements ... 22
5.2.2 Legal requirements ... 26
5.2.3 Other requirements ... 29
6 Design ... 31
6.1 Overall design ... 31
6.2 Implementation of requirements ... 34
6.2.1 Component monitoring and integrity ... 34
6.2.2 Recovery ... 36
6.2.3 Denial of service ... 37
6.2.4 Auditing ... 38
6.2.5 Chain of custody ... 38
6.2.6 Labelling and Integrity of collected data ... 39
6.2.7 Time synchronization ... 40
6.3 Communication ... 40
6.3.1 Communication protocol ... 41
6.3.2 Message types ... 41
6.3.3 Communication requirements ... 44
6.4 Components ... 45

6.4.1 Collection coordinator 45
6.4.2 Collection agent 51
6.4.3 Log server 55
6.4.4 Collection monitor 57
7 Implementation 58
7.1 Communication 58
7.1.1 Communication protocol 58
7.1.2 Message formats 59
7.2 Collection coordinator 60
7.3 Log server 60
7.4 Collection agent 60
7.5 Adapters 60
7.5.1 Pipe adapter 61
7.5.2 Snort adapter 61
7.5.3 Run adapter 61
7.5.4 Auditd 61
7.6 Portability 62
8 Future work/Research directions 63
8.1 Evaluation 63
8.2 The design 63
8.3 Related research directions 65
8.3.1 What to log 65
8.3.2 Report format 66
8.3.3 Enforcement of security properties 66
8.3.4 Standard for logging support in applications 66
8.3.5 Standard logging module in operating systems 66
8.3.6 Reduction of false positives 67
8.3.7 Isolation of attacked nodes 67
8.3.8 Analysis of the collected data 67
9 References 68
Appendix A User documentation 71
A.1 Installation 71
A.2 Usage 71
A.3 Format of the configuration files 72
A.4 Implemented adapters 72
A.5 How to write your own adapter 73

1 Introduction

This chapter presents the background of the project presented in this report as well as a brief introduction to the problem area of the project and the relevant technologies, intrusion detection and computer forensics. Lastly the hypothesis that the project is based on is presented and explained.

1.1 Project background

The Norwegian Research Council announced an extra grant within the IKT 2010 program, for projects within security, during the spring of 2002. The grant focused on selected themes and application areas. The project we received funding for and that is presented in this report is categorized under the theme enforcement of security properties and the application area network infrastructure.

The project is motivated by two needs that we have identified, the need for a system that simplifies and automates the collection of forensic evidence and tools for enforcement of security properties.

1.1.1 Collection of forensic evidence

Today hackers are seldom made accountable for their actions. However, physical intrusions are almost always reported and investigated. There are many reasons for this difference, like that the knowledge of how to investigate a computer intrusion is generally low, many organizations do not want computer intrusions to be publicly known, and the collection of forensic evidence from computer systems is often a very troublesome and time consuming activity.

Security work today is highly centred towards protective measures like access control, risk analysis and patching of security vulnerabilities. In addition in recent years detection measures have reached widespread use through Intrusion Detection Systems (IDS). Computer systems are as buildings protected with access control and intrusion alarms, but computer intrusion are seldom reported to legal authorities while physical intrusion most often are.

The attackers have an advantage over the security personnel. They can choose the place and time of their attacks, having only to discover and exploit one vulnerability, or at most a limited number of vulnerabilities. While, security personnel must protect all possible points of attack. This leads to a sever inequity in the technical and economic resources required for defence versus those required for offence. We believe that if attackers are made accountable for their actions through criminal prosecution or civil lawsuits, it can provide the deterrent effect that is needed to even the playing field.

Work is needed in many different fields to make effective investigation of computer intrusions possible. One area where work is needed is in the legal field, especially in agreements between states, since the Internet knows no national boarders. Such work is underway in the *G8 Recommendations on Transnational Crime* [30] and the *Council of Europe Convention on Cybercrime* [31]. Work is also needed in improving the technical

ability to track and identify the attacker. A survey of the current state and the challenges within the field of tracking and tracing of Internet-based attacks is presented in *Tracking and Tracing Cyber Attacks: Technical Challenges and Global Policy Issues* [32].

This project addresses another important aspect of investigation of computer intrusions, the collection of computer forensic evidence at the attacked site. Today the collection of forensic evidence is a very troublesome and time-consuming activity.

"There is a lack of guidance to employees as to how to respond to intrusions and capture the information required to conduct a law enforcement investigation." [33]

"Great progress has been made in recent years to speed up the re-installation of the Operating System and to facilitate the reversion of a system to a 'known' state, thus making the 'easy option' even more attractive. Meanwhile little has been done to provide easy ways of archiving evidence (the difficult option)."[15]

Today logging is conducted in different formats and there is often no deep analysis behind the events being logged. In any case it's seldom that the analysis has taken evidentiary concerns into account. Another part that is missing in today's logging mechanisms is handling of the collected data so that it fulfils the requirements that are put on it to be judged admissible in a court of law.

The project presented in this report is founded on the observation, that if networks are equipped with a system for collecting forensic data during intrusions, that handles the collected data in a way so that the legal requirements are fulfilled and present the collected data in a standard report format, the collection of forensic evidence would be greatly simplified and more accessible.

1.1.2 Enforcement of security properties

When security is discussed, three themes are often addressed: confidentiality, availability and integrity. One example is ISO 17799 [10]. An important aspect that often seems missed is how implemented security measures can be verified to work as expected. This is what is known as enforcement of security properties. Deviations from the expected behaviour can be caused by a combination of faults in the configuration and faults in hardware or software of the security mechanism.

Periodical risk analysis can provide enforcement of security properties, but a risk analysis is only performed perhaps once or twice a year – or more seldom. Penetration testing can also be used to find shortcomings, but not on a daily basis.

Protective security measures like encryption, access control, and firewalls include in themselves limited possibilities to uncover breach of other protective security measures. During a breach of the local security policy the failing security mechanism(s) cannot be trusted to report authentic information. The consequence is that the breach cannot be guaranteed to be detected by the security mechanism itself. In order to "benchmark" the

implemented security solutions on a daily basis, a tool that collects data related to events that reveals breaches of the local security policy is needed.

The data that needs to be collected to enforce security properties may differ from the data needed to collect from an evidentiary perspective, however the same collection mechanism can be used. Data collected with the aim of enforcing security properties are also not subject to the rigorous requirements that forensic evidence is subject to. However, one should note that data that are collected with the aim of enforcing security properties might still end up in a court of law.

The project presented in this report has so far focused on the data to collect from an evidentiary perspective. Issues related to enforcement are therefore deferred to future work, and is further discussed in section 8.3.3.

Tools for enforcing security properties can have many applications: It gives system administrators' possibilities to discover whether implemented security solutions actually do their intended work or to investigate a security breach. From the internal IT-revision's viewpoint it enables them to find the reason behind the security breach so that improvements can be proposed and responsibilities placed. Management will have many of the same needs. They too want to place responsibility. Not necessary to make disciplinary actions against employees, but for economic reasons. If a security solution installed by a third party fails to fulfil a contract, compensation could be claimed. It is also possible that software vendors for e.g. operating systems will be held responsible for security flaws in their systems one day.

We also recognize the need for other tools that support enforcement of security properties, like tools that continually scan or searches the network for the software components present. This is especially important for networks where the right to add new software has been distributed to the users of the network. New unwanted software may have been added with malicious intent or in good faith. However, it's important for the security personnel to have information about all new software present, since every new service represents a possible attack target. This type of tool and others for enforcing security properties have however not been worked on so far in the project.

1.2 Relevant technologies

1.2.1 Intrusion Detection Systems (IDS)

There are two main categories of IDS, misuse based and anomaly based. IDS are further classified after from where they get their input. Network Intrusion detection Systems (NIDS) get their input from the network, network packets, and Host Intrusion Detection Systems (HIDS) get their input from data available on the individual hosts.

Misuse based systems search for known threats (coded in signatures), e.g. known bit-strings in network packets or know sequences of local system events. A signature might be: search the network packet's payload for a string that equals assembly code for opening a shell on a Unix system. Such a string could be used in a buffer overflow attack.

Anomaly based systems search for unusual behaviour, such as a PC receiving large amounts of traffic at late hours. The information to search might be found in hosts' system logs or in statistics captured from the network. With detection based on anomaly detection there may be a rather long delay before the attack is detected. This is caused by the fact that an anomaly is often only detected after some threshold has been reached. For example in the case an alert is triggered after x messages (satisfying some criteria) have been received the alert is first generated after x-th packet, but the attack may have been initiated with the first packet.

1.2.2 Computer Forensics

There are two main types of computer forensics analysis tools. The first focuses on capturing and analysing information on the network, while the second focuses on the same tasks locally on hosts. Based on the discussion in various books like [5] and mailing lists [3],[4], it is our conclusion that most emphasis within the computer forensics community focuses on how to collect evidence on a host after a compromise and after the system has been powered down. This is sometimes termed collection on a corpse system. The opposite collection before the system has been powered down is called collection on a live system.

One can argue that the forensic process begins when a new system is put in production. A forensic examination will depend upon the choices done at this stage in the lifecycle of a system. If no logging is enabled or if not the right events are logged, the forensic data possible to collect after a compromise will be considerably reduced.

In the following two sections a general introduction to forensic analysis tools will be given.

1.2.2.1 Network Forensic Analysis Tools (NFAT)

These tools capture and inspect packets on the network. Example of such tools, include netDetector [1] and SilentRunner[2]. They can log the complete traffic flow, just log part of the packets or those meeting certain defined criteria. When the packets are saved, they can be analysed later as part of the forensic process. One can consider NFATs as a surveillance camera for computer networks.

Some of these tools aim at making it possible to replay the traffic flow on the network, enabling investigators to discover exactly what happened and when. Others are tuned for capturing interesting statistics and chosen packets or headers. The information gathered is analysed by specially designed applications that search for e.g. packets sent to a host or occurrence of some bit-string.

1.2.2.2 Host Forensic Analysis Tools (HFAT)

These tools mainly follow the same general idea: binary copy the hard drive of the affected system to a safe place using the least intrusive techniques on the affected system. If a full forensic analysis was started on the affected system, the state of the system would be altered and useful information lost, as well as excluding the analysis as viable in a court of law. Copying can be done using the *dd* command on a Linux system or with specialized software

like EnCase [7]. Once copied, the safe copy is hashed and possibly signed with a digital signature to ensure integrity for possible later use in a court of law.

The safe copy is examined with tools like EnCase [7] and The Coroners Toolkit [8]. Interesting information on the system could be logs of activity, files of a special kind (e.g. pictures) or containing information of interest. It is possible to search through the files registered in the operating systems file table, or for hidden or recently deleted files on areas of the disk marked as unused or corrupt. Activity on the host is – assuming logging is enabled – stored in various files. Depending on the level of details in logging, past activity can be examined.

There are also tools for collection on a live system. The most famous being The Grave Robber that is included in The Coroners Toolkit [8]. The Grave Robber can for example collect information like: the image of every running process that is removed from disk and the command history.

1.3 Hypothesis

In our application to the Norwegian Research Council [34], we presented the following hypothesis:

"The hypothesis is that a module based distributed system where every module is specialized at collecting forensic evidence in a specific environment will simplify the collection and increase the quality of forensic evidence. The collection will be trigged, refined and coordinated based on the intrusion detection information supplied by IDS technology. In a large network it is not possible to continuously collect all the data needed for forensic analysis due to performance and the enormous amount of data. The approach we propose circumvent this problem by collecting fine-grained data only when needed and only at the hosts and network devices affected by the attack."

1.3.1 Concept scenario

A system implementing this hypothesis will depend upon IDS technology to start logging of data. The system collects fine-grained data, which is detailed and hence requires a lot of space to be stored. Since storage space is not infinite and collection of the data might consume considerable resources, it could cause a bottleneck e.g. on a web-server. A preferable solution would be to collect data only when a security incident is suspected.

The system could be compared to a surveillance camera. Instead of recording a stream of pictures the system collects detailed information about the events that occur in the network. It can collect both system events on the hosts and the packets traversing the network. If this data collected is detailed enough the state transitions of the network could be recreated given the initial state of the network. The system could offer the same playback functionality as a surveillance camera and even the possibility to zoom in and out (adjusting the level of detail).

IDS offer the possibility to detect when a security incident is under way. If we know the signature of the attack, we can use a signature based IDS (NIDS or HIDS) to alert a

specialized collection module when an attack is launched. This module can then start collecting the data and store it, preferably on a dedicated server.

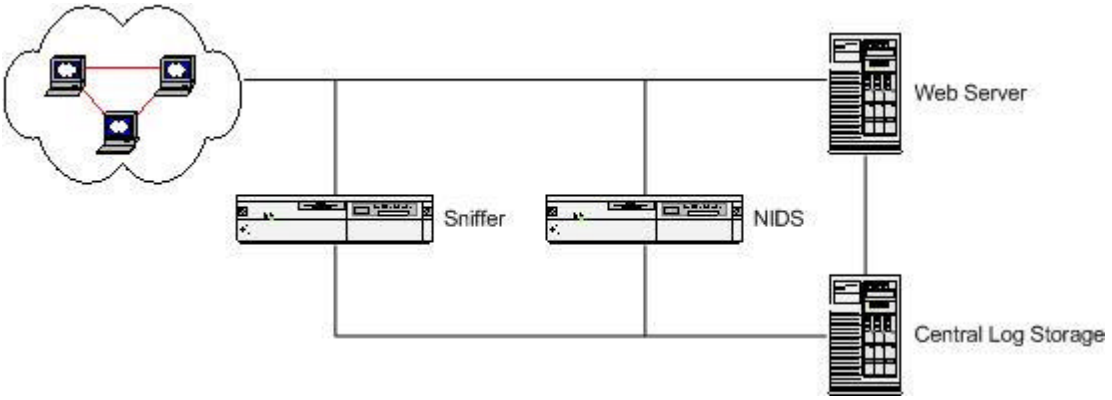

Figure 1: Simple scenario

The figure above illustrates a simplified scenario. The Sniffer functions like a NFAT systems, it has the ability to collect the packets traversing the network. When the NIDS alerts that an attack is underway, the Sniffer is informed of it and it starts to collect the packets traversing the network and transfer them to the log server. The alert from the NIDS also triggers a message to the host under attack, in this case the Web Server, telling it, it should start fine-grained collection. The data collected locally on the Web Server is also sent to the Central Log Storage.

Note, that a system implementing the hypothesis does not have to use only one IDS but that it can use many IDSs and that they can be both misuse and anomaly based.

1.3.2 Scope

The work presented in this report focus on automated collection of forensic data. Including both an analysis of which data to collect from a forensic perspective as well as how this data generally should be handled for it to be admissible in a court of law.
Issues related to the analysis of the collected data and the report format of the data is not treated in this report.

1.3.3 Research challenges

Developing a working prototype based on the hypothesis raises some important research challenges. We have chosen to focus on some of them. These research challenges are presented below, and discussed among others in later sections of the report.

- What types of data is valuable as forensic evidence?
- What are the requirements on a secure distributed collection tool?
- Which requirements must the collection process meet for the collected data to be viable forensic evidence?
- How can these requirements be implemented?

1.4 Structure of the report

The remainder of this report is organised as follows:

The next chapter, chapter 2, presents prior work related to the work presented in this report.

Then in chapter 3 some fundamental and current imperfections in intrusion detection technology are presented, which affects a system implementing the hypothesis.

Chapter 4 discusses which data to collect from an evidentiary perspective. The criteria that data has to fulfil to be suitable for collection by a system implementing the hypothesis are also presented.

The following chapter, chapter 5, treats the requirements that have to be put on a system implementing the hypothesis.

Note that much of the material presented in both the chapter 4 and 5 are applicable to all logging or collection of forensic evidence. Not just collection conducted by the system proposed in this report.

Chapter 6 presents the current design of the system proposed in this report.

In chapter 7 a prototype implementation, implementing parts of the design, is presented.

During the project, new research directions have been identified. These and work that we didn't have time to carry out with in this face of the project are presented in the last chapter "Future work and research directions", chapter 8.

2 Related work

There are different opinions on whether IDSs should be used to collect and preserve forensic evidence [42]. However, the approach we propose only uses the intrusion information IDSs generate to coordinate the collection of forensic evidence. The collection and preservation is conducted elsewhere. Consequently, the approach we purpose does not impose any additional collection and preservation tasks on IDSs. The information gathering and preservation is done outside the IDS, with the logs that it generates as one of many information sources.

Work on combining forensics and intrusion detection and their roles in the overall network security picture, has been done by Stephenson [44]. Tom Perrine and Abe Singer in [43] states the need for incident management tool that interoperate, scale and decrease security investigator workload.

We have not found any papers or research projects describing anything similar to the concept we propose. During the cause of the project Psionic released (in June 2002) a tool named ClearResponse [22]. Psionic (later bought up by Cisco) claims that ClearResponse offers an automated, just-in-time analysis of each targeted host to determine whether a compromise has actually occurred. When ClearResponse receives an alert from the local NIDS it launches an analysis on the affected host to determine if the attack was successful or not. It checks in order: the operating system that runs on the targeted host, the patch level of the targeted host, and lastly it analyses system information including registry entries, system files, log files etc. ClearResponse does not use remote agents on the hosts. It uses the existing authentication infrastructure to get read-access to the targeted hosts. If the analysis confirms an attack, Psionic claims that the system collects forensic evidence and copies this information to a secure location. It's unclear what information is collected. It should be noted that the design goals of ClearResponse and the system we propose differs. ClearResponse is designed to lower the investigative burden of the NIDS operator. The system we propose aims at simplifying and automating the collection of forensic evidence. We have not been able to test the tool and since ClearResponse is proprietary software we have not been able to study its design and implementation.

In [39] a first responders tool for evidence collection on a live system is discussed. It has similarities with the system we propose, as it can be seen as an automatic first responder's tool. Another tool for collection of forensic data on a live system is The Grave Robber included in The Coroners Toolkit [8].

Many sources discuss the possibilities for active response from intrusion detection systems when an attack is detected. Active responses often mentioned include: tracing back the intruder to the source of the attack, terminating the session carrying the attack, and direct a firewall or router to block traffic from the attacker. All these active measures have their problem, mainly that they can be used in denial of service attacks. Some sources also mention the possibility to trigger collection of system data. However, the aim of the data collection does not have to be collect forensic evidence, it can also be to refine the detection

capability. Below are a number of citations related to starting data collection as a active response to an IDS alert:

"Active response can trigger further collection of data by starting another application…" [46]

"Thus, our overarching objective is to be able to generalize a theory that supports intrusion detection and forensics computer science in the same system, regardless of its specific application. We believe that this overarching objective presents some significant challenges in the case of host based intrusion detection systems. Most of these challenges have to do with defining the lengths to which we wish to extend forensic analysis of the target system. For example, capture of a physical image of a computer disk by the IDS (or an extension of the IDS) presents some significant difficulties." [42]

"IDS agents communicate in "agendas" that carry suspected intrusion intentions based on the threads given by the global IDS. This allows local agents to predict possible attack directions and fine tune its own system specific auditing utilities to pro-actively look for all relevant data. Later on, when meaningful data is observed, local IDS agent reports back sub-goal confirmation as well as the supporting data." [47]

"Permanently recording the vast quantity of information flowing across a large network is, in practice, impossible. However, if a network IDS has identified a signature suggesting an intrusion, it could selectively record the relevant packets for subsequent legal examination. Thus, an IDS could greatly reduce the volume of data needed for legal support. Unfortunately, given the false alarm rates of current network ID systems, it may be difficult to make a strong case for evidence collected this way. In some cases evidence might simply be missed, and in other cases much irrelevant information might be collected. Whether such information would hold up in a court of law remains to be seen." [9]

3 IDS issues

The hypothesis that is the basis for the work presented in this report leaves a lot of questions unanswered and problems left to solve if a system is to be constructed with the hypothesis as a basis. This section will address problems related to the system's use of intrusion detection technology. Possible solutions are also discussed. The problems and their solutions are further treated in later sections of the report.

3.1 False negatives

Current IDS technology does not detect all attacks, called false negatives in IDS terminology. This implies that the collection of fine-grained data will not be activated for every attack, since the IDSs will miss attacks. The data collected under this approach can therefore not be that important that it must be collected during all attacks. Data that are that important shall be collected continually. Even if IDS technology continually improve it is unlikely that IDS technology in the foreseeable future will be able to detect all attacks.

The incompleteness of IDS technology, misses to detect attacks, has many fundamental causes, including:

- A NIDS does not have a complete picture of the network and hosts it monitors. A packet might for example not be accepted at the destination because some particularities in the implementation of the TCP/IP stack of the destination host. The NIDS do not know this and accepts the packet. It may of course also be the other way around the destination host accepts a packet that the NIDS reject.

- Misuse based IDSs use signatures to detect patterns that indicate an attack is under way. The IDSs may not have any signature for the attack in question installed, there might not be any signature for the attack available yet, or it may have one or many installed for the attack but none of them detect the attack.

- Many NIDSs are not capable reconstructing higher-level protocol traffic. They often just analyse single IP packets.

3.2 Collection delay

Between an IDS detects an attack and the nodes affected by the attack start fine-grained logging, some time passes. During this time span plus the delay caused by the analysis in the IDS no fine-grained collection is active. The attack gets a head start. This problem is particularly large for some anomaly based IDSs. For example if the IDS examines the traffic flow to different ports and generates an alert if the traffic to some port is un-normally large, an eventual attack may have gone on for some while before the alert is generated. The delay also depends on the attack type. A slow scan takes longer time to detect than an attempt to exploit a buffer overflow vulnerability.

This loss of data might not be a problem for some types of data. As an example, information about what files are currently open might not be so sensitive to this delay if one assumes that files usually are open longer time than the delay. For other types of data that are sensitive to

the missed collection during the delay, the collection can be activated continually but the local buffer were the data is saved is overwritten with a period larger then the delay. With this technique no data needs to be missed if the delay is smaller then the period used. The performance advantage of fine-grained collection is lost but storage and network resources are still preserved.

3.3 Denial of service

An attacker can attack an IDS by generating a lot of packets or system events with the sole purpose of triggering alerts. The goal of such an attack can for example be denial of service or to hide the real attack in all decoy alerts. Since a system implementing the fine-grained approach acts upon alerts, the effect of the attack will be amplified if not countermeasures are implemented by the system. A number of such countermeasures are presented in section 6.2.3.

3.4 One attack is possibly detected as many attacks

Intrusion detection systems may generate many alerts for each attack. These alerts can be grouped together during analysis. However, methods for grouping or correlation of alerts and the data collected as a result of the alert are not treated in this report.

4 Analysis of which data to collect

This chapter will discuss the importance of collection of different types of data from a forensic perspective. How different types of data fit into the hypothesis (see section 1.3) will also be examined. This discussion will not be complete as this subject alone is enough for a report of its own.

What is not logged is lost. Every application and operating system on every device on a network represents a logging opportunity. During analysis of an intrusion one can never have too much information as stated by Marcus Ranum in [36] *"I've done numerous incident responses in my career. In every one, I wished I had more information."* As long as the information is hierarchically structured, more information can only be positive. That is the analysis tool should support a birds view of the data and the possibility to drill down in detail. However increased data collection comes with a price, which is consumption of resources. The fine-grained collection approach tries to limit these negative effects.

From Goal-Oriented Auditing and Logging, By M. Bishop, C. Wee and J. Frank [37]:
"With respect to computer security, logs provide a mechanism for analyzing the system security state, either to determine if a requested action will put the system in a non-secure state, or to determine the sequence of events leading to the system being in a non-secure (compromised) state. If the log records all events that cause state transitions, as well as the previous and new values of the objects changed, one can reconstruct the system state at any time. Even if a only a subset of this information is recorded, one might be able to eliminate some possible causes of a security problem; what remains provides a valuable starting point for further analysis."

Today only a subset of the events that lead to state transactions are recorded. This can have many causes like, performance and storage issues, the question about what to log has not got the attention it deserves, and more data complicates the analysis.

We have only found one paper analysing what to log [37]. The paper contains a study of prior work up to 1996. According to the authors none of the listed prior works treat the question of how to determine what to log in any depth.

Again from [37]:
"In the past, scant attention has been paid to the mechanics of developing a theory of how to determine what should be logged; the focus has instead been on the mechanisms."

The technique presented in [37] derives from the security policy which data to log. The security policy is viewed as a statement of constraints upon commands and states. Violation of these constraints must be audited. This combined with the system model gives the low–level commands and data that must be logged to enable effective auditing.

The focus in [37] is auditing to determine violations of the security policy. This guides which data to log. Our focus is somewhat different. We are not only interested in detecting security violations, but also to get a detailed picture of the events after the violation.

The Orange Book [38] specifies what events should be logged and what information the log should contain for each event. However, we are not aware of any work that motivates the choices made regarding which events to log. The Orange Book specifies four classes of criteria namely A, B, C, and D, with systems meeting the highest criteria (A) providing the best level of security assurance. There are a number of subdivisions in classes C and B. Classes C2 to A1 require the ability to log security relevant activates on the system. For example, criteria C2, requires that the system must log the following events: use of identification and authentication mechanisms, introduction and deletion of objects into a user's address space (e.g., file open, program initiation), actions taken by computer operators and system administrators and/or system security administrators and other security relevant events. Moreover, for each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event, origin of request for identification/authentication events and name of object for object introduction and deletion events.

4.1 Analysis

It's natural to sort the information about an attack chronological, in a time line of the actions taken by the hacker. Therefore a timestamp should be added to all events logged. The time line can be seen as a recording of the attack. As with a video recording of a traditional burglary one first of all wants to have an as complete picture as possible and one wants the ability to zoom into details.

All data that can shine light on the actions of the attacker are valuable, especially such data that can be used to identify the attacker. A commonly used analysis technique is to attempt to link the 'fingerprint' of the case with other known cases. Characteristics of an attack that make up the 'fingerprint' include, time of attack, vulnerability exploited, script or method used in the exploit of the vulnerability, and the action taken after compromise. The attacker might have created new user accounts with the same username, executed the same sequence of commands, and modified the same configuration files etc. during separate attacks. This type of correlation between attacks can uncover information that was previously overlooked.

The attacker tools used and the actions taken by the attacker indicate how well connected and skilled he is. Another important aspect of the analysis is to try to figure out the purpose behind the attack, examples include: espionage (motivated by information on the system), use the comprised system as a springboard for new attacks, and using the compromised system as a "hacker's clubhouse".

4.2 Data suitable to be collected under the fine-grained hypothesis

[39] Divides fragile evidence into three categories:

- Transient data – Information that will be lost at shutdown, such as open network connections, memory resistant programs, etc.
- Fragile data – Data that is stored on the hard disk, but that can easily be altered, such as last accessed time stamps.
- Temporarily accessible data – Data that is stored on the disk, but that can only be accessed at certain times.

Transient data can include data kept in memory to speed up access, programs that are running but have been removed from the disk. This technique can be used by a hacker to hide a malicious program like a sniffer or backdoor. In addition, transient data can include information that in it self is transient as information about ongoing network connections, content of received network packets, users logged on etc.

Fragile data are the data stored on the hard drive but that can easily be altered. For example file access timestamps, temporary files, and swap space. Once this data has been altered there is no way of recovering it.

Temporarily accessible data include encrypted data that is only available in plain text format during the times it is decrypted, for example after a password is presented. The actual password used to decrypt the data is also temporarily accessible data.

All three of these types of fragile evidence are appropriate for collection by a system for collection of data on a live system, including a system implementing the fine-grained approach.

In addition it should not be essential that the data, as discussed in section 3.1, must be collected during all attacks, since the IDS functionality that the system depends on will miss attacks, so called false negatives. There is also a delay from the attack is staged until the collection is started, as discussed in section 3.2. For types of data that are sensitive to the missed collection during the delay, the collection can be activated continually but the local buffer where the data is saved is overwritten with a period larger then the delay. In this way all data since the attack was staged is still available when the collection is started.

The data could also based on the reasons below be inappropriate to collect continually:

- Performance
- Storage
- Network
- Privacy

The tree first reasons are all related to resource consumption and they are all related to the amount of data collected. More CPU power, storage and bandwidth are consumed as the size of the collected data increases. The consumption of network resources is an effect of that it is assumed that the data is sent to a central log server for security reasons. Performance problem could also arise if the collection of the data is in itself resource

consuming, for example if the collection infers a lot of disk access or advanced aggregation of a number of different sources of information.

Storage and network resource consumption can be addressed through compression. However, compression increases the CPU resource consumption.

The data collected might intrude on users' privacy. This risk increases if the data is very detailed and/or sensitive. The data might for example represent chat conversation. If such sensitive data should be collected continually it would severely affect users' privacy even if strict access control and encryption would be used. By only collecting this type of information during presumed attacks the risk for misuse is somewhat reduced. If the attacked node is a personal workstation the user could even be alarmed that fine-grained collection is ongoing to even more reduce the potential privacy intrusion. The privacy can also be improved by pseudonymizing the data, i.e. exchange user names etc. for pseudonyms or cryptographic hashes. An authorized authority can later personalize the data again. Privacy vs. Intrusion Detection Analysis by Lundin and Jonsson [40] presents how data can be pseudonymized.

4.3 Collection candidates

The state of a computer system is made up of the image of the operating system, the processes running in memory, and the files on the system. Therefore if one can collect the information about modification of these entities (including creation of new files and execution of new processes) and the time of these modifications one can record all state changes. Modification of operating system image is done through loadable kernel modules or recompilation of the kernel.

Data that can be collected on a computer system can come from three different sources: the operating system, the applications and the network. From the operating system a myriad of different data can be collected, like information about resource usage, file access etc. The data that can be collected from an application depends on the nature of the application; a firewall application offers different possibilities then a web server application. From the network the traffic traversing it can be collected. In this context the network is the medium transporting the information. The routers on the networks are seen as nodes with operating system running a router application.

Data from all three sources give valuable information and data should be collected from all three. They all contain complementary information. As an example the plaintext of encrypted network traffic can only be collected after it has been decrypted by the receiving application. The network packets in turn contain information that might not be available from the other sources, like for example the address of the sender or information of how far away the sender are (deduced from the time-to-live field).

Infrastructure applications like, firewalls, IDS, DNS, routers, proxy servers, LDAP servers, DHCP servers, dial-up servers, VPN etc, are all sources of valuable forensic data.

Information that can be collected from the operating system, include:

- Resource usage statistics (e. g. CPU, memory, hardware availability)
- Process information (e.g. active processes, active DLLs, user executing process)
- File system information (e.g. file access, file permissions)
- Network information (e.g. open ports, traffic statistics)
- User accounting information (e.g. new user added, login, logout)
- System calls executed

Many of the types information mention above can be used to extract the same information. As an example, file access and system call information can give information about the processes executed. However, from a forensic viewpoint the value of the evidence will improve, as several different sources indicate the same event.

The data that can be collected from an application and OS is determined by the implementation. One can only hope that the logging offered has been based on a thorough analysis. However, some applications and operating systems may provide the possibility to add modules or plug-ins that can extract data. For example Linux offers the possibility to extend the logging facilities with loadable kernel modules.

4.3.1 System calls

The only way applications can use the services the operating system provides is through the system calls that the operating system offers. So if information about all executed system calls with their parameters and return values are collected one gets a complete picture of the state transitions of the system.

There are different types of system calls. Not all system calls can help an intruder gain full access to a system. But they might be "useful" for an attacker in other ways, like when launching a Denial of Service attack. Bernaschi et. al. [11] have analysed the system calls used in the Linux kernel 2.2.

In [11], the system calls have been grouped in categories according to their functionality, as presented in Table 1.

In addition, they classified each system call according to Table 2, which portrays the essential features of the four levels of threat they consider.

Group	Functionality
I	File system, devices
II	Process management
III	Module management
IV	Memory management
V	Time and timers
VI	Communication
VII	System info
VIII	Reserved
IX	Not implemented

Table 1 System call groups

Threat level	Description
1	Allows full control of the system
2	Used for a denial of service attack
3	Used for subverting the invoking process
4	Harmless

Table 2 Threat levels

The classification in Table 2 "corresponds to a threat hierarchy, since a system call classification at threat level *n* may be employed also to carry on an attack at threat level *m* if $m \geq n$. For example, if a system call allows the attacker to gain access to a privilege shell, then the attacker has full control of the system (threat level 1). In this case it is trivial to carry on a Denial of Service attack (threat level 2)" [11]. The attacker now has the ability to e.g. shut down a process.

The system calls qualifying for threat level 1 are the following:

Group	System Calls
I	open, link, unlink, chmod, lchown, rename, fchown, chown, mknod, mount, symlink, fchmod
II	execve, setgid, setreuid, setregid, setgroups, setfsuid, setfsgid, setresuid, setresgid, setuid
III	init_module

Table 3 System calls qualifying for level 1

An interesting result seen in Table 3 is that only system calls within group I, II and III can be used to gain full control of the system.

4.3.2 File access timestamps and file content

File access timestamp is an example of fragile data, that is data that is stored on the hard disk, but that can easily be altered. The problem with file access timestamp is that every time a file is accessed, the last timestamp is overwritten. The history is lost and it's impossible to construct an accurate time line of the file accesses.

An attacker can easily modify the timestamps of the files he/she has modified so that it looks like they have not been accessed during the attack, since the history of access is not saved. It is however not enough to just to save the history locally, since then the attacker can modify it too. It must be stored remotely.

Most operating systems offer at least three time stamps for each file. They are called modification, access, and change of status times. Access time indicates the last time the file was accessed or opened for content viewing, modification time holds the last time the content of the file was modified, and change of status time contains the time of the last change of the meta information about the file, such as permissions and ownership.

In addition to collect the information that modification has happened the actual change to the file can be collected or the whole file if it is being deleted.

With the capabilities of today one might see that an executable or script named X was executed at some time but one cannot be certain about what this process was doing since the original X might have been replace with a new executable named X or X may have been deleted. If all eventual changes to X where collected and stored remotely this would not be a problem anymore.

Not changes to all files need to be collected if resources are scarce. Some files are more critical then others, like password and trust configuration files, and some may contain more interesting information then others, like temp directories. Temp directories serve as a scratch pad and working directory for the system. Files in temp directories are normally more volatile the other files. These temporary files can give important information about the state changes of the system.

5 Requirements

A complete list of all requirements that can be put on the system is not discussed here. The basic system functions of the system are presented and system attributes from the categories security requirements and legal requirement. Selected system attributes from other categories that are seen as important are also presented. Requirements related to operation, management, installation etc are not handled.

Ultimately the legal authorities determine the requirements on a system for collecting forensic evidence. Projects like the Computer Forensic Tools Testing (CFTT) [41] that is to establish a methodology for testing computer forensic software tools by development of general tool specifications, test procedures, test criteria, test sets, and test hardware will likely also influence the requirements put on computer forensic tools. The requirements presented below have different importance; some are essential for the collected data to be admissible in any court of law while others shall be seen more as desirable. It should be noted that legal requirements differ from nation to nation. The requirements presented below shall therefore be seen as general requirements that do not necessarily apply to all nations.

5.1 System Functions

The system has three main functions:

- The system shall collect forensic data.
- The system shall compile the data into a standard report format.
- The system shall aid in analysing the collected forensic data.

The system functions define the high level basic functionality of the system. The focus so far in the project has been on collection and not on analysis and definition of the standard report format. A discussion about which data to collect is presented in chapter 4. The analysis capabilities of the system and the standard report format will be defined in a later phase of the project.

5.2 System Attributes

System attributes are characteristics or dimensions of the system; they are not functions. The attributes have been categorised into security, legal and other requirements.

5.2.1 Security requirements

5.2.1.1 Integrity

In the following discussion the concept of integrity has been divided into access, process and data integrity.

- *Process integrity.* Asserting that the process, in this case the collection system, has not been exposed to malicious or accidental alteration.
- *Data integrity.* Asserting that data, in this case the collected data, has not been exposed to malicious or accidental alteration or destruction.
- *Access integrity.* The prevention of modification or destruction of an asset by an unauthorised user or entity.

For the collected data to be of any evidentiary value the access integrity must not have been violated and process and data integrity guarantees that an eventual violation of access integrity is detected. Since it is paramount that the authenticity of the collected data can be trusted all three types of integrity must be implemented to the highest degree possible.

The access integrity of the data can be violated during several stages. The source of the data can be modified to deliver false data. The process that collects the data can be modified to deliver false data. Once collected the data can be modified while it is being transferred and stored. And lastly the data can be modified while in permanent storage.

Access integrity is implemented through access control mechanisms, that guarantees that only authorized users or entities can modify the source of data, the process collecting the data, and the actual collected data. Data integrity is implemented by cryptographic techniques that detect modification of the protected asset if the protection barrier offered by access control is broken. Process integrity can also make use of the same cryptographic techniques as data integrity and in addition monitoring techniques (e.g. heart beats).

However, neither access control or cryptographic integrity techniques can protect against a hacker that has gained super user privileges. The super user on a host has access to everything on that host. Implying that the victim's own hard drive(s) is not a safe place to store the collected data. Another reason for not storing on the victim's hard drive(s) is that we don't want to overwrite what could be crucial evidence. It is therefore essential that the collected data is transferred and permanently stored on a separate dedicated host, so that the hacker do not have access to data collected prior to the point of compromise.

The source of data is either some application or the Operating System (OS). The trust placed in these applications and the OS must be carefully evaluated. The security mechanism of the OS and the installation and configuration of the application in question must assure that only a user with super user privileges can modify the source's executables. If an attacker acquires super user privileges on the host that the data is collected on it cannot be guaranteed that the hacker to cover his own actions does not modify the source application or OS. The modification can cause the stream of data generated by the source to leave out the data the attacker wants to hide or even worse the source can have been changed to cause serious damage. Such damage could include deleting files, corrupting files, transferring information to a host controlled by the attacker etc. It's therefore desirable to be able to control the integrity of the source, so that if it is violated an informed decision can be made if the source still should be used or not.

As long as the host on which data is collected from is not compromised the collected data must be protected against alteration during the stage from collection to permanent storage. If the host has been compromised the hacker can modify the collection process. As with the source it would be desirable to be able to detect eventual modification of the collection process.

To achieve this detection capability the system must include components that control the integrity of the data sources, the collection processes and the communication and storage process.

The collection process can be designed so that the hacker must terminate the collection process to alter its behaviour, the termination of the collection process can then be detected. To increase the protection of the collection process it should be as hard as possible to terminate the collection process. It would be desirable that the collection process only can be terminated after a reboot.

The data integrity of the collected data must be protected during transportation from the host where it was collected to the dedicated host where it is stored. The communication must provide guaranteed delivery. The components communicating must authenticate each other, so that protection against replayed data and false data can be accomplished.

Finally while in permanent storage the access and data integrity of the collected data must be uphold. The protection of the access integrity should be as strong as feasible, taken into account the cost of the access control solution. The combination of physical and logical access control must at least implement the lowest level required by the legal authorities.

The integrity of the software components of the system must be verified manually during installation and periodically during operation. The integrity verification during operation can be handled automatically, but should also be checked manually to protect against corruption of the automatic integrity verification tool. The result of the verifications should be documented and protected according to the same requirements as the data collected by the system.

The integrity of the system configuration must also be protected. On restart when the configuration is reread it must be confirmed that the configuration has not been modified by any unauthorized entity.

It should be documented under which conditions the integrity of the collected data cannot be trusted. For example data collected after an attacker has acquired super user privileges cannot be trusted. These conditions must be clearly defined so that it can be determined if the collected data is trustworthy or not by an operator of the system.

These requirements imply that it is impossible or hard for a hacker to modify the data collected before the time of compromise, since the collected data is stored on a separate dedicated host. To modify data collected before the compromise the dedicated hosts must also be compromised and if the data is stored on write once medium not even that is enough. Then the hacker must physically access and destroy the collected data.

5.2.1.2 Confidentiality

Confidentiality is the prevention of unauthorized disclosure of information. The need for confidentiality of the collected data is motivated by the requirement of privacy for users - which actions are being logged, that the collected data may contain sensitive information

about the computer environment of the organization, and that the collected data may contain sensitive information that would harm the investigation if they where known to the public. Therefore the confidentiality of the collected data should be protected during transport between the components of the system and in permanent storage.

5.2.1.3 Access control

Access control must be implemented to fulfil the confidentiality and access integrity requirements. Access control to the source of data (application or OS), the collection process and the collected data are dependent on the underlying access control mechanism in the used OS. Super user privileges should be required to access these resources. The confidentiality of the collected data can further be protected by access control mechanism above the OS level.

To increase the strength of the authentication mechanism the system can use one of or a combination of password (some thing you know), token (something you have) and biometrics (something you are). The level of strength required must be determined by the requirements set by the legal authorities.

5.2.1.4 Availability

With availability we mean the protection of the system from denial-of-service threats that might impact system availability. The fault tolerance requirements discussed below will also limit the effect of denial-of-service attacks.

The system shall be able to recover from system crashes, either accidental or caused by malicious activity. Upon start-up, the system must be able to recover its previous state and resume its operation unaffected.

The system shall provide graceful degradation of service. If one component goes down, the rest of the system should work as normal.

As discussed in section 3.3 is intrusion detection technology inherently vulnerable to denial of service attacks, since it's possible to generate packets or system events with the sole purpose of triggering alerts. Since the system reacts on the alerts by starting data collection, the system will amplify the effect if not countermeasures are included in the design. Such countermeasures should be implemented.

5.2.1.5 Fault tolerance

The system is dependent on the functionality of IDSs. Since the IDSs used are not part of the system, it can only reduce the dependency by using many IDSs if possible.

It's desirable that that all functionality in the system is implemented with redundancy or other techniques that ensure that the system will offer full functionality even during a failures of single components.

The system shall support out of band (SMS, email etc) notification of serious system functionality degradation to the operator of the system, so that the functionality can be restored manually.

5.2.1.6 Logging and Auditing

The system must log its configuration, all action taken internally and all interaction with outside systems and users. The logging should be at a sufficiently low level so that the state of the system can be recreated. The confidentiality and integrity of the logs must be protected.

The system shall include support for auditing the logs of the system. It's also desirable that the system includes functionality for automatic auditing of the logs.

5.2.1.7 Protection of credentials

Communication confidentiality and integrity requires that the components posses credentials. These credentials must be protected against unauthorized access.

5.2.2 Legal requirements

This section will try to answer the question about which requirements that has to be put on a data collection system if the collected data should be accepted as digital evidence in a courtroom.

One can divide collection of forensic data into two categories depending on the state of the system that the evidence is being collected from. If the system has not been rebooted or shutdown since the attack took place the collection is taking place on a live system. The opposite is collection on a corpse, a system that has been rebooted or shutdown since the attack took place. When collecting on a live system one wants to obtain as much volatile data as possible, as for example memory content, state of network connections and state of running processes. This data is lost when the system is powered down. On a corpse collection traditionally takes the form of making a binary copy (image) of the disk, which is further analysed. We will focus on legal requirements on systems that collect data on live systems. However, much of the requirements discussed will be applicable to both types of system.

The International Organisation On Computer Evidence (IOCE) [35] defines digital evidence: *"Information stored or transmitted in binary form that may be relied upon in court"*. What requirement the phrase *"may be relied upon in court"* implies may differ from nation to nation. The IOCE was appointed just to draw up international principles for the procedures relating to digital evidence, to ensure harmonisation of methods and principles among nations and guarantee the ability to use digital evidence collected by one state in the courts of another state. In this section we focus on common requirements on digital evidence that are used in most nations.

According to [15] digital evidence must be:

- Admissible: It must conform to certain legal rules before it can be put before court.
- Authentic: It must be possible to positively tie evidentiary material to the incident.
- Complete: It must tell the whole story and not just a particular perspective.
- Reliable: There must be nothing about how the evidence was collected and subsequently handled that casts doubt about its authenticity and veracity.
- Believable: It must be readily believable and understandable.

According to [33] evidence must satisfy two tests: admissibility and weight. Where weight implies that the evidence *"must be understood by, and be sufficiently convincing to the court"*. Weight encompasses authenticity, completeness, reliability and believability.

Evidence can be judged inadmissible for a number of reasons depending on the laws of the individual nations. Examples are hearsay and that evidence has been acquired unfairly or illegal, for example without a search warrant. Once evidence has been ruled admissible the court can assess it for weight.

The remainder of this section will present requirements that a system for collecting digital evidence must fulfil for the collected data to be judged admissible and of weight.

The Chain of Custody must be carefully maintained. The Chain of Custody is the process of documenting the complete journey of the evidence during the life of the case. The Chain of Custody is used to prove that what is presented in court is what was originally collected.

The Chain of Custody includes answers to the following questions:

- Who collected the data/evidence?
- How and where?
- How was it stored and protected in storage?
- Who took it out of storage and why?

Anyone who has possession of the evidence, the time at which they took and returned possession, and why they were in possession of the evidence must be documented. Every action performed during analysis of the data should be carefully documented, so that the actions can be repeated resulting in the same analysis result.

Since we here discuss a system for collecting digital evidence, the chain of custody requirements regarding handling of collected evidence during analysis is not applicable to the system. The actions taken during the collection process must however be documented. Answers to questions like the following must be answered:

- The machine the data was collected from (identified with serial number and/or hostname)?
- The source on the machine that generated the data?
- The part of the system that collected the data?
- The time interval of the collection (including the time zone used) with as high resolution as available and feasible?
- The quantity/size of the collected data.
- Which other parts of the system handled the data on the way to permanent storage?
- The name and version of the system used in collection?
- The configuration and state of the system during the collection?
- The reason behind the collection (alert that trigged collection)?

Traditionally the records that make up the chain of custody have been handled manually. However, these records can be created by a computer system. As with the collected data should the Chain of Custody document also be signed to protect its integrity.

Every collected item must be identified and labelled. The label should include:

- Incident identification
- Item identification
- Brief description of the item
- Date and time of label creation
- Signature of label

The pair incident and item identification uniquely identifies the item.

The system should make the data collected for each incident, the collected data with labels and the Chain of Custody document, available for analysis.

To further protect the integrity of the evidence (including the collected data and the documentation of the collection) should the data be signed with multiple hash algorithms, so that the integrity is secured even if a cryptographic attack is discovered against one of the algorithms used.

As stated in [33] and [16], the raw data collected should always be preserved, even if the data is processed in order to make it easier to understand, perform analysis or transform it to a standard format. The original data may not be modified and therefore should all analysis work on a copy of the original data. Defining what's raw data on a computer isn't a trivial matter. The alerts from an IDS can be considered to be derived data, the raw data is the dump of the traffic on the network that triggered the alert. The requirement must be that the data collected should be as raw as possible. For example, if one has the possibility to collect the raw network traffic in addition to the alerts the IDS generates it is certainly positive.

For data that is derived from raw data the credibility of the system that derived the data is essential. The same holds for every system handling the digital evidence. If the system is closed, that is the source code, architecture and specification is not available for review the admissibility of the evidence can be questioned. The legal authorities must sanction a closed system (data that the software collects or produce are approved as evidence) or it must be defendable by expert witnesses for the data collected to be of any value in court. However, it may be difficult for an expert witness to be credible in defending a closed system. If he does not have access to the implementation details of the system his testimony has limited value and if he has access it can be claimed that he has dependencies to the system owner. Further it is problematic if the parts in the trial have different access to the details of the system implementation. With open systems both parties in a legal dispute can examine the system and both parties will have access to the same information.

In any case is it essential that only systems or tools that are properly evaluated and validated are used. The same of course also holds for all techniques and procedures that are used during the collection and forensic examination. Validation includes that the technique's or

procedure's limitations have been identified, it has been demonstrated fit for its purpose, it has procedures in place for monitoring performance and it is documented and the results obtained are reliable and reproducible [18].

It is an advantage or even a requirement to have many independent sources of data indicating the same event. As stated in [33] *"Single streams of evidence are unlikely to be adequate to convince a court; what is required are multiple independent streams of evidence which corroborate each other."* To be able to link these streams together it's important that time synchronization is used and that it is accurate. Other sources of information can be local on the system, for example logs of network traffic, but it can also be logs from intermediate hosts that the attacker relayed through during the attack.

Commands executed during data collection on a live system may modify the state of the system and thereby taint the evidence. They can for example modify access times to files, introduce new log entries and modify the memory content. If possible execution of such commands should be avoided. If not, their execution needs to be clearly documented so that the state changes they infer can be separated from the ones introduced by the attacker.

However, when collecting data on a computer system it's inevitable that the state of the system is modified at lower levels. For example processes (including the process collecting data) modify the RAM during their execution and if too much RAM is used it is a risk that the use of virtual memory overwrites data on disk that might hold crucial evidence. The collection process will change the state of the system, but the changes must be kept to a minimum.

To limit the effect of state changes introduced by the collection process on the collected data, data that might be modified when other data is collected should be collected first. As an example, when a file is accessed the access time of the file is changed. Therefore the access time of a file must be collected before the actual file is accessed and collected.

The collected data and the documentation of the collection should not be stored on the same system as the data is collected from. Storage on physically separate media is required, media on another system than the one compromised. Traditionally the collected data is sent to and stored on a centralised log server. One centralised point of storage is easier to secure.

The access to the data both physically and logically must be strictly controlled. No one that really doesn't need to have access to the data should have it. If it can be shown that numerous individual possibly can have accessed the data its value as evidence diminishes. The defence attorney can argue that anyone who had access to the evidence could have altered it. He/she does not have to prove that the evidence was really tampered with for this tactic to work. He/she only has to point at the possibility.

5.2.3 Other requirements

The architecture must be extendible, meaning that it is easy to and documented how to add support for new types of data sources and log formats both regarding collection and

analysis. This is critical since new operating systems and applications is continually developed and the system has to be able to plug in these new sources with as little work as possible.

The architecture should also be scalable, meaning that it should be possible to add collection capabilities to additional nodes as the need arises.

The system must allow dynamic reconfiguration. That is it must be possible to change the configuration of a part of the system without taking the whole system down.

The architecture and implementation should aim at being as independent of underlying software (e. g. operating systems, databases etc.) as possible.

The system should avoid starting any new processes, open any files, open any sockets or pipes in conjunction with starting collection locally, since this will increase the chance of the system being detected by the attacker. The system should be as silent as possible to not arise suspicion.

As discussed in section 3.2 there is a delay from the time of the attack to the collection is started locally. This delay should be minimized as much as possible in the design.

6 Design

The following chapter discusses the design of a system architecture that is capable of starting fine-grained collection of log-data as an answer to an alert from an IDS as discussed in 1.3, and that satisfies the requirements presented in the last chapter. The design presented leaves some design issues unresolved, but it discuses possible solutions to these issues.

The main function of the system is to collect forensic evidence after a security incident has been detected. The system should be able to collect evidence on a number of platforms, and from many different sources. The design should be modular; it should be easy to extend to support new kinds of targets and new types of analysis or processing of the data.

6.1 Overall design

The system consists of a number of server components and a client component on every node where collection is to be conducted. The server components may be co-located on the same node. However, the log server component should be located on a dedicated node according to the requirements.

The components represent functionality. Fault tolerance (traditionally implemented through redundancy) of this functionality is currently not addressed by the design and is deferred to future work. However, the design is aimed at simplifying introduction of fault tolerance through: simple design, components with clearly defined responsibilities, and state-less components.

The client components connect to targets on the node they are located. The targets are the processes that actually generate the data that are collected by the system. Examples of targets include the OS, system logs, like syslog on Unix systems, application logs, programs that generate output that tells us something about the system, like for example ps(1) on Unix systems, and intrusion detection systems. Any program that generates, collects or extracts information can be a target.

The information that the targets generate is collected by a collection agent. The main function of the collection agent is to be the point of communication between the local targets, and the rest of the system. The communication between the collection agent and its targets are handled through adapters, one for each target (see Figure 2). The collection agents also monitor its targets through the adapter for the target, if the adapter and target supports monitoring. Some targets may only be possible to monitor through periodically monitoring that the target process is still alive. More advanced targets may offer hooks that make it possible to alert the collection agent of events like, restart, shutdown, failure etc.

The adapter component fills the function of enabling support for new targets without having to change the collection agent implementation. To the collection agent every target presents the same interface through an adapter. Besides forwarding the data from the target to the collection agent, the adapter has a number of additional functions. The adapter may translate the data from a target-specific format to a general system wide format; it may provide

capabilities to instruct the target of which data to collect; and it may offer monitoring of the target.

The data collected by the collection agents are sent to the log server, the central server where all the collected data from all the different targets in the system is stored. The log server also stores the logs generated by the system.

The collection coordinator is the component that directs the collection. It receives alerts from the IDS(s) in the system, analyses these, and issues the orders that make the appropriate collection agent(s) start data collection.

The design also includes two components that support analysis functionality of the collected data, the log analyser and the management console. The management console lets an operator configure the system and analyse the collected data. The log analyser conducts automatic analyses of the data in the log server that are presented through the management console and possibly given as feedback to the collection coordinator. The detailed design of the log analyser and the management console is however deferred to future work.

The collection monitor monitors the system. It compiles information about the system's state from the status, heartbeat and failure messages it receives from the other components. The system's state can be monitored through the management console and the collection monitor can be configured to alert an operator through mail, SMS etc.

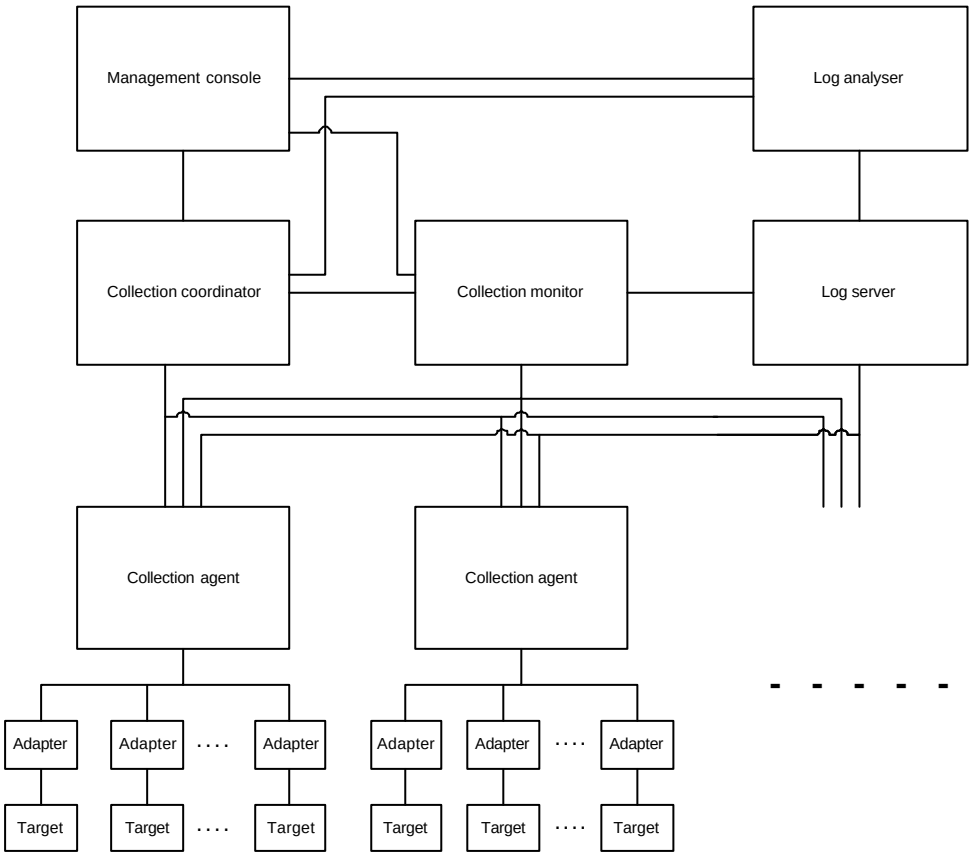

Figure 2: An overview of the proposed design of the system.

The following scenario is an example of a possible chain of events:

1. An attack is detected and one or many alerts are generated.
2. The alert(s) is routed through respective adapter to its collection agent.
3. The collection agents send the alert(s) directly to the collection coordinator.
4. The collection coordinator generates collection orders based on the alert(s) and distributes them to the appropriate collection agents.
5. The collection agents instruct the affected adapters according to the collection order.
6. The adapters instruct the targets to start collecting data according to the order, and send the collected data to the collection agent. Alternatively, the collection has been activated the whole time but the data has not been relayed to the collection agent until now.
7. The collection agents forward the collected data to the log server.

During the execution of this scenario the collection coordinator may receive new alerts that results in new collection orders and maybe revocation of old collection orders. In a system that gives feedback, it's possible that the collection coordinator also can generate collection orders on basis of information that it receives from the log analyser, or based on information from other parts of the system, like the collection agents.

Note that the collection agents may be configured to have some collection activated regardless if any collection order has been received. Data collection activated constantly is termed background collection.

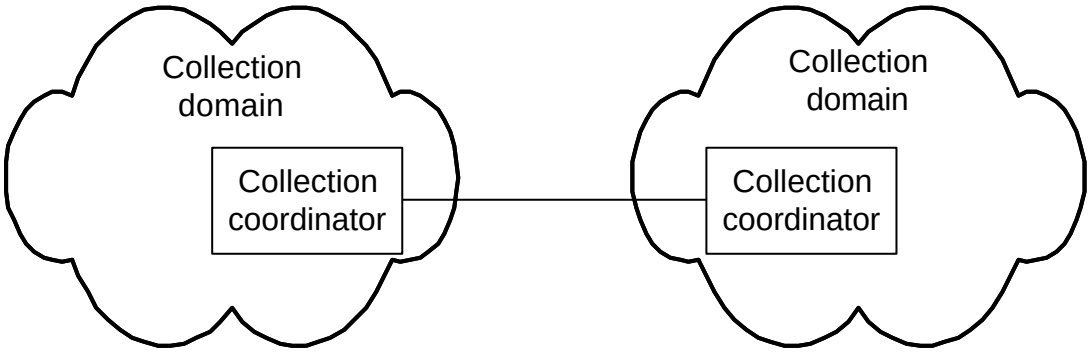

Figure 3: Communication between collection domains

Figure 3 illustrates a collection domain. Each collection domain is controlled by one collection coordinator and consists of the collection coordinator, the collection monitor, the log server and the collection agents and targets connected to them.

A large network can be organized into several domains and an organization that is geographically distributed may have one collection domain at each site. To follow distributed attacks there is a need for communication between these domains. Such communication is

done between the collection coordinators in the respective domains (see Figure 3). The details of this communication are deferred to future work.

6.2 Implementation of requirements

This section discusses the implementation of some of the requirements presented in chapter 5.

6.2.1 Component monitoring and integrity

It's critical that the data collected are authentic. The data collected on a host an intruder has gain super user privileges on cannot be trusted, but it's important that it's as difficult as possible for the intruder to modify the collected data without being detected. It's also important to identify which data can be trusted, and which that cannot be trusted. The system must be able to detect disruption of the components running in the system, and the integrity of the configuration files and the component executables.

The surveillance functionality of the system is implemented by the collection monitor. Each component of the system send heartbeat messages (periodic messages that indicates that the component is alive), status messages at status change (for example at start, stop, deterioration of service level etc.) and failure messages to the collection monitor. The status and failure messages are also sent to the log server. The collection monitor controls that the heartbeat messages, status and failure messages are correct and that the heartbeat messages are received as expected. The period between successive heartbeats is determined by the configuration of the system.

Failure messages are sent if a component detects a malformed packet, it may for example contain an invalid signature or sequence number. A failure message is an indicator to the collection monitor that the sender of the malformed packet is corrupt.

Like all other messages, the heartbeat message contains an identification of the sender, a sequence number, and an encrypted/signed hash of the message that protects the integrity of the message (See section 6.3.2.4 for more details on the content of the heartbeat message).

The heartbeat mechanism offers a limited protection against modification of the component sending the heartbeat. If the hacker will replace the component with a corrupt component he has to retrieve the key used to sign the heartbeat messages, and the next valid sequence number, and he must time the replacement so that the heartbeat period is uphold. When the collection monitor receives a heartbeat it calculates the time since the last heartbeat. If this period differs significantly from normal, an alert is triggered. The time of a normal period is based on the configured period and statistics on normal deviations from that period. How this is implemented is deferred to future work. The collection monitor sends all calculated heartbeat periods to the log server where they are stored so that they are accessible during analysis.

An attacker can however change the behaviour of a component, without being detected by the heartbeat mechanism, through modification of the underlying libraries and operating system services that the component uses. For example in Linux an attacker that has acquired

super user privileges (root) can modify the behaviour of the OS by loading kernel modules into the kernel. To reveal the loading of malicious kernel modules, the system should collect data about kernel module loading events via its ordinary collection mechanism. The system can acquire some protection against malicious libraries if the libraries are statically linked into the components' executables. The integrity of the system's executables is protected by manual and automatic integrity checks (see below).

The adapters monitor the targets to the degree allowed by the target. Some targets may only be possible to monitor through monitoring the liveliness of the target process. Others may offer hooks to connect to, e.g. SNMP trap. The adapters inform their collection agent of status changes to their targets. The collection agents compile the status information received from their adapters and send status and heartbeat messages to the collection monitor, and status messages to the log server.

The adapters are executed in the same process as their collection agent. Hence, the collection monitor monitors them too. It can be made harder to disrupt the service of the collection agents and their adapters if they can only be shutdown after a reboot. Then an attacker that wants to replace a collection agent must do so by activating the corrupt replacement on another node, since the time to reboot is considerably longer then a reasonable heartbeat period. Automatic restart is discussed further in the next section.

The events that should generate an alert to the administrator of the system are determined during configuration. Obvious candidates include failure of any component and especially the log server and the collection coordinator, and if the collection monitor receives a failure message indicating the corruption of one of the components in the domain.

The log server should for security reasons not use the production network to send messages. Instead, the log server communicates with the log analyser, the management consol and the collection coordinator through a dedicated administrative network. The log server sends heartbeats over this network to the collection monitor.

Each component's configuration is stored in one or several local configuration files. A copy of the configuration is also stored in the collection monitor. This implies that every time a local configuration file is modified a copy of it must also be installed in the collection monitor. In the status messages sent at start-up and restart, the local configuration is included. The collection monitor compares this configuration with its copy of the configuration. If they differ, an alert is generated. The reason that the complete configuration is included in the status message and not just a hash of the configuration is that the configuration is also sent to the log server, to make it available during analysis.

The integrity of the software components (executables) for the system must be verified during installation and periodically during operation. The integrity verification during operation can be handled automatically, but should also be checked manually to protect against corruption of the automatic integrity verification tool. The automatic integrity verification could be done either periodically or after every access to the monitored file. The result of the verifications should be documented and protected according to the same

requirements as the data collected by the system. How the result from the automatic integrity check is documented and stored is currently not specified in the design. However, it would be natural that the result is transferred to the log server like the rest of the system logs and to the collection monitor.

Collected data cannot be trusted after a system compromise where the attacker has acquired super user privileges or after the collection monitor has detected possible corruption of the system. However, note that it's not guaranteed that the collection monitor detects an eventual system corruption or that an eventual detection is correct.

The trust to put in the collected data must be ultimately determined by analysing the collected data, and the system's message exchange and logs, for signs of system compromise. This data can reveal if and when an attacker has acquired super user privileges. If so is the case the collected data, even if the attacker generates or modified it, will give valuable information. One cannot however be sure that it indicates what has really happened.

The collection monitor can detect system corruption in many ways, heartbeat periods significantly different from normal and through failure messages and status messages. The correctness of the collection monitors detections must be assessed during analysis, by examining both data collected by the system and the system data logged.

6.2.2 Recovery

The requirements section states that the system should recover from system crashes. Upon start-up, the system shall recover to its previous state and resume its operations unaffected.

Currently, the system components do not maintain any state information. Implying for example that if a collection agent goes down, it will not resume the collection activated before the crash, except the background collection. The introduction of state is deferred to future work. If state is introduced, the integrity of the state must be protected.

Automatic restart of components makes the system more fault-tolerant, and more difficult to stop. Automatic restart requires that the components are highly reliable so that they do not go down repeatedly because of faults in the component, and thereby jeopardize the system's stability.

Automatic restart can be implemented in a number of ways. One-way is to have a daemon or a kernel module (Linux) monitor the process. However, with both these methods a hacker with super user privileges can disable the daemon or kernel module and then terminate the component. It would be desirable that the components only can be terminated after a reboot. That is a setting in a configuration file is changed so that after the next boot the component is not active any more. It's unclear whether any current operating systems support this feature.

Automatic restart implies that keys used by the component to authenticate itself must be stored unencrypted or encrypted by a password, which then must be stored unencrypted. The last key or password in the chain must be stored unencrypted.

Automatic restart should be done locally, and not via the network. This is important, since remotely controlled restart would be a big security hazard. If an intruder could restart a component without having access to the node that the component is running on, he/she would be able to direct a denial of service attack on the remote component through repeated restart orders.

6.2.3 Denial of service

As discussed in section 3.3, intrusion detection technology is inherently vulnerable to denial of service attacks, since it's possible to generate packets or system events with the sole purpose of triggering alerts. Since the system reacts on the alerts by starting data collection, the system will amplify the effect if not countermeasures are included in the design. Possible countermeasures include:

- Analyse the alerts and data collected to determine if extra logging should start. Simple thresholds can be used. If too many alerts are received within some interval, the fine-grained collection is restricted.

- Implement a threshold based on local resource consumption, so the fine-grained logging cannot consume more resources then specified by the threshold. This threshold must be carefully selected, so that it's not reached under normal conditions.

- Dynamically adjust the level of fine-grained collection. When resource usage is high, collect data at a higher granularity. For example if data about open files is gathered periodically, the load locally can be used to determine the period.

- Analyse the effects of the eventual attack locally on the attacked node to determine if collection should stop, because the alerted attack is a false alarm. In section 6.4.1.5 it is further discussed how to determine when collection should be stopped.

These counter measures can be implemented locally by the collection agents, by the collection coordinator or by a combination of them both. The prototype implementation does not currently implement any of these countermeasures, since this issue needs to be investigated further.

In any case the collection agents should not be configured to collect more data then feasible from a resource perspective. That is when maximum collection is activated the resources consumed should not remarkably effect the ordinary applications running on the node. It is currently not known if the resource consumption of the collection generally is of the degree that it may influence normal operation.

The log server must be scaled so that it can handle if the maximum fine-grained collection is activated in all collection agents. The only way to verify this is to test the installation before deploying it.

6.2.4 Auditing

The requirements state that the system must log its configuration, all action taken internally, and all interaction with outside systems and users. The logging should be at a sufficiently low level so that the state of the system can be recreated. The confidentiality and integrity of the logs must be protected

All messages between the components of the system except heartbeat messages are always sent to the log server. Heartbeat messages are only sent to the collection monitor. The collection monitor processes heartbeat messages and send the result to the log server.

The components' logs should be transferred to the log server. The analysis of what the individual components should log and the mechanism to transfer them to the log server is deferred to future work.

The confidentiality and integrity of the system log should be protected by the same mechanism as the data that the system collects.

6.2.5 Chain of custody

The requirements that the documentation of the chain of custody sets on the system are discussed in section 5.2.2. They can be summarized as follows: the origin of the collected data and its way from collection to storage in the log server must be documented.

The alert that triggered the collection order and the collection order itself are both sent to and stored in the log server. Every collection order contains a unique identifier that is used to link collection order, and data collected due to the collection order, together.

After start-up and restart, the collection agents send their configuration, including information about the adapters they use and the configuration of the adapters, and the hostname of the node to the log server. Every adapter type is identified by a unique id, which is made up of the name of the author of the adapter, and the name and version of the adapter. The collection agent software is identified by its version and the ids of the plug-ins included in the collection agent. Plug-ins are described in section 6.4.2.1.

Every collection agent has a unique id assigned to it in its local configuration file. It's the administrator's responsibility that no collection agents are assigned the same id. However, in the case of duplicate ids the collection monitor will detect it. The collection agents assign locally unique ids to its adapters. The collection agent id and the adapter id together form a domain unique identifier.

To all data collected, the adapters add the time interval when the data was collected, the format type, and the quantity of the collected data.

The collection agent adds more information to the collected data. It adds the time it received the data from the adapter, the id of the adapter and its own id and the ids of collection orders that caused the start of the collection.

6.2.6 Labelling and Integrity of collected data

The requirements state that every collected item should be labelled and that the label should include:

- Incident identification
- Item identification
- Brief description of the item
- Date and time of label creation
- Signature of label

The label is constructed by the log server from information in the log messages it receives. The incident identification is created by combining the collection order id that trigged the collection and the domain unique identification of the collection coordinator that sent the collection order in question.

The item identification is made up of the sequence number that all messages contain combined with the unique identification of the component that sent the message. Every component has a separate sequence number counter for each message type. The components reset the sequence numbers at restart.

The requirements state that the Chain of Custody information and the collected data should be signed to protect their integrity. These signatures and the sequence numbers should not be confused with the signatures and sequence numbers used to assure communication security, as described in section 6.3.3. The communication security mechanism assures that messages originate from authorized components. The signatures discussed here assure integrity and trace ability of the collected data and the data that describes the state of the system.

Collected data should be signed by the collection agent that collected the data and the log server if it is received correct, i.e. the communication security mechanism validates that it originates from an authorized component. The content of all messages should likewise be signed by the sending component and the log server, as all messages are stored in the log server.

The keys used to sign the data must be managed. The public keys corresponding to the private keys used to create signatures must be stored so that they can be accessed during analysis and their integrity must be protected. At the same time the secrecy of the private keys must be protected. It needs to be further investigated how the signature keys should be managed (i.e. updated, stored and secured) and which cryptographic algorithms that should be used.

Note that the components do not verify the signatures and sequence numbers described here during operation. The components only verify the signatures and sequence numbers that

implements communication security. If a message contains an invalid signature or sequence number the component sends a failure message. The signatures and sequence numbers described in this section is used to ensure the integrity of the data during analysis.

6.2.7 Time synchronization

To be able to correlate events on different machines, and to make the evidence more reliable it's necessary with some form of time-synchronization between the different components. The system uses the Network Time Protocol (NTP) [14]. The official Coordinated Universal Time (UTC) should be used to avoid problems with correlating timestamps from different time zones.

NTP is based on a multi tiered system where each layer is called a stratum. Servers at each level peer with each other and provide time services to lower levels. Servers at the top or in stratum 1 are directly connected to atomic clocks or radio based time receivers. By compensating for their distance from the authoritative time sources, these receivers provide highly accurate time services to stratum 1 servers.

Below stratum 1, NTP servers are supposed to obtain time from servers above them as well as at their own level (stratum). The configuration instructions say that each top level server within a specific domain should be a client to at least two servers at the level directly above it and peer with all the other servers in their own domain at their same level as well as at least one other outside peer on the same level. The servers receive time from but never provide time to the servers at the next higher stratum. Peers receive time from and provide it to other peers. There should be at least three coordinated top-level servers in each domain so each network should communicate with at least six outside servers at the next higher (numerically lower) stratum and at least three outside peers in the same stratum.

If available, it's desirable that the timeservers in the collection domain connect directly to a primary reference source, usually a wire or radio clock that is synchronized to a radio station that provides a standard time service.

Measures to protect the integrity of the timeserver and the communication between the timeserver and the components are deferred to future work.

6.3 Communication

The requirements section states that the communication between the components in the system must guarantee delivery and offer both integrity and confidentiality. These requirements apply to both inter-process communications within a host and network communication between components on different hosts.

This section focuses on the network communication between components on different hosts. Communication between the collection agents, collection coordinator, log server and collection monitor will be treated, while communication with and between the log analyser and management console are referred to future work. Communication between adapter and target is presented in detail in the section 6.4.2.3.

6.3.1 Communication protocol

All the information that is sent over the network is pushed; the sender initiates the communication. The collection agents push information to the collection coordinator, the log server and the collection monitor, while the collection coordinator pushes orders to the collection agents, etc.

6.3.2 Message types

Figure 4 gives a summary of the different message types and where they're sent. Detailed descriptions of the message types are given below.

	Receiver			
Sender	Collection Agent	Collection Coordinator	Collection Monitor	Log Server
Collection Agent		• Alert	• Status • Heartbeat • Failure	• Log • Alert • Status • Failure
Collection coordinator	• Order		• Heartbeat • Status • Failure	• Status • Order • Failure
Collection Monitor				• Status • Failure
Log Server			• Heartbeat • Failure	

Figure 4: Messages that are sent between the different components

Figure 5 shows the message header that is used for all types of messages. *Ver* is short for protocol version. *Type* is type of message. *Format* denotes how the message shall be interpreted depending on the *Type*. For alert and log messages the *Format* denotes how the payload is formatted. For status messages the format denotes the kind of status message, while the value is ignored, and should be zero for heartbeat and order messages. *Payload length* gives the total length of the payload. The *Sender Component Type, Sender Component ID, Plug-in ID* triplet uniquely identifies the origin of the message and the pair *Receiver Component Type* and *Receiver Component ID* uniquely identifies the receiver. The *Component Type* denotes the kind of component, e.g. if the node is a collection coordinator or a collection agent. The *Component ID* field contains the identification of the

component. The *Component Type* together with *Component ID* uniquely identifies the component, which sent the message. *Plug-in ID* is used to identify the plug-in or adapter that originally generated the message. This field should be zero if the message not originates from a plug-in or adapter. The two last fields in the header gives the timestamp of the message with microsecond precision.

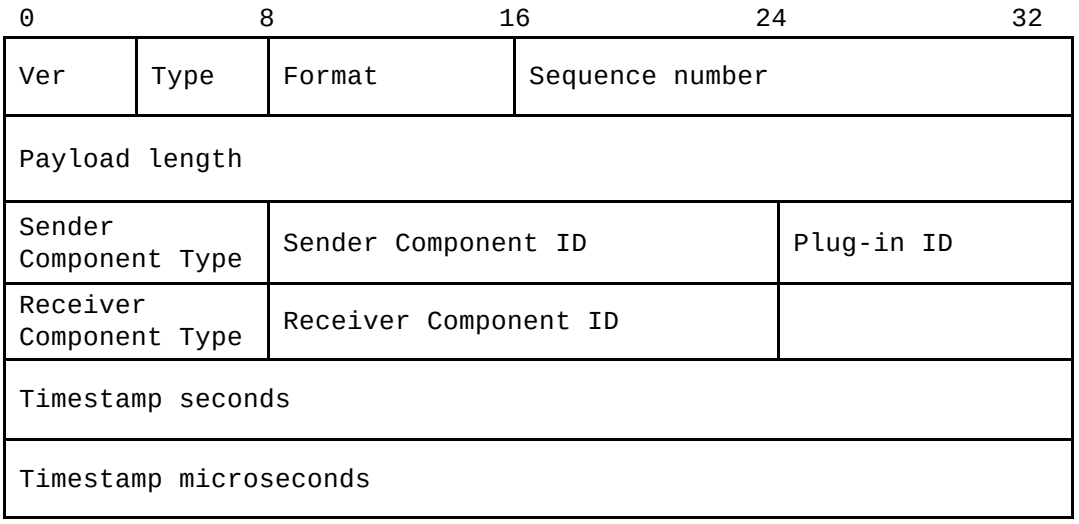

Figure 5: Message header

6.3.2.1 Alert message

A collection agent that has received an alert from one of its targets sends an alert message. The message is sent to the collection coordinator and the log server. The origin of the alert is a target of a collection agent with IDS-functionality or alternatively, a system component with attack detection capabilities. An example of such a component could be the log analyser that might detect attacks when analysing the logs. In the future, the collection agents may also include attack detection functionalities themselves and just not through their targets.

The alert message contains the alert and the header presented above. The time stamp in the header identifies when the alert was issued. The alert format id that is located in the header is used to identify how the data that the alert consist of is formatted. This identifier is globally unique and every alert format supported by the system must be assigned such an id.

The processing of alert messages by the collection coordinator is discussed in section 6.4.1.

6.3.2.2 Log message

The log messages contain collected data that is sent to the log server for storage. They contain information about events on the network and the hosts supervised.

It's still an open question if these messages should be in a standard format, or if they should be stored in their original format. Presently the system has support for both.

In addition to the header, the message must contain the log record and two time stamps. The two time stamps identify the interval in which the adapter collected the log record. The log format id is filled into the format field in the header and is used to identify the format of the

log record. This identifier is globally unique and every format supported by the system must be assigned such an id.

In addition, the log message should contain the ids of the collection orders that triggered the collection of the log record in the message. However, note that if normal background logging caused the collection of the log record, the message won't contain any collection order ids.

6.3.2.3 Status messages

Status messages are sent by the components in the system to inform about state changes. All components send status messages to the collection monitor, which checks these messages, and triggers an alarm if some part of the system doesn't seem to work as it's supposed to, or if there are indications of unauthorized modification of the system. The status messages are also sent to the log server so the state of the system is available during analysis.

A collection agent sends four types of status messages: start, stop, collection state changed, and deterioration of service. The start status message contains the configuration of the collection agent, and a list of adapter ids. It also contains information about the software version of the collection agent and the ids of the plug-ins used. A 'start' status message is sent when the collection agent is started or restarted. The 'stop' status message contains the cause of the shutdown. A 'collection state changed' status message is used to indicate the set of fine-grained collection activated. It is sent after a collection order is received. The message contains the id of the collection order that trigged the message and list of adapter ids, with their adapter type id and their configuration. A deterioration of service message is sent out when one adapter detects that a target no longer fulfils it collection assignment. The deterioration of service message contains the adapter id, adapter type id and the configuration of the adapter of the target that has failed.

The collection coordinator and the log server send two types of status messages: start and stop. The 'start' status message contains the configuration and software version of the collection coordinator or log server and is sent at start up. Similar to the 'stop' status message of the collection agents the stop message of the collection coordinator and the log server only contains the cause of the shutdown.

The collection monitor sends status messages to the log server for storage. As the other components it sends 'start' and 'stop' status messages. In addition, the collection monitor sends all calculated heartbeat periods for the other components in the domain and information compiled from the status messages it receives in 'system state' status messages to the log server.

As stated above all components should send a status message at shutdown. Components should catch the signal (e.g., POSIX or ANSI signal) that indicate request of shutdown or restart, and sends a status message before acting upon the signal. However, not all signals can be caught. In those cases, the component will be terminated immediately, and it will have no chance of sending a status message.

6.3.2.4 Heartbeat message

Heartbeat messages are messages that a component sends periodically to indicate that it's still operating. All heartbeats from all components are sent to the collection monitor, where the time since the last heartbeat is calculated. If this period differs significantly from normal, an alert is triggered. The collection monitor sends all calculated heartbeat periods to the log server where they are stored so that they are accessible during analysis.

6.3.2.5 Collection order message

Collection order messages are messages sent from the collection coordinator that tells the system to start or stop fine-grained collection and what collection to start. These messages are discussed in detail in section 6.4.1.

The collection orders contain an id, the collection order id, which is used to link collected data to the collection order that trigged the collection.

6.3.2.6 Failure message

If a component detects a malformed packet, it may contain an invalid sequence number or an invalid signature etc, it sends a failure message to the collection monitor and the log server.

6.3.3 Communication requirements

The requirements section states that the communication between the components in the system must guarantee delivery, offer integrity and confidentiality.

The confidentiality of log, heartbeat and failure messages must be protected. However, alert and order messages should perhaps not be encrypted because of the extra delay introduced. These messages carry on the other hand sensitive information. Messages between collection coordinators in different domains should be encrypted, since they may traverse the Internet.

The integrity of all message types should be protected. All message types should also be protected against replay.

The design does not specify how the communication security requirements are implemented. Communication security is provided by the communication protocol used by the systems components, like BEEP or HTTPS (see section 7.1.1).

However, the system manages the credentials needed for secure communication. Therefore each component must be configured with the identity and credentials of all components it will need to communicate with.

- The collection agents must be configured with the identity and credentials of the collection coordinator, collection monitor and log server in the domain.
- The log server, collection monitor and collection coordinator must be configured with the identity and credentials of all other components in the domain.

The components must also protect their credentials from access by unauthorized entities.

Each component must also assure that it is informed by the underlying communication protocol of authentication errors so that it can act upon them.

It is not currently defined if symmetric or asymmetric cryptography is to be used by the system to secure the communication. If an asymmetric solution is chosen it must be decided if certificates (PKI) should be used, or if the public keys should be manually registered in the different components. In the long run probably a certificate solution is preferable. It provides easy administration, since a new collection agent can be added without having to register the new collection agent's public key in all components and the other component's public keys in the new collection agent. With a certificate based solution the administrator only has to issue a new certificate to the new collection agent.

6.4 Components

6.4.1 Collection coordinator

The collection coordinators basic responsibility is as the name indicates to coordinate collection. It is accomplished by receiving input, processing this input and finally sending out collection orders addressed to the collection agents in the domain of the collection coordinator. The collection orders contain information about the collection that should be started.

It's important that the process of generating a collection order from received input is as fast as possible to minimize the delay from the point of detection to when data collection starts.

The mapping from input to collection order should be easy configurable. New attacks are discovered and new signatures are generated continually. The administrator of the collection domain must also be able to customize the set of mappings in use in the domain.

6.4.1.1 Inputs

The collection coordinator in a domain can receive input from a number of different sources:

- Alert messages from the collection agents in the domain.
- Feedback messages from the log analyser in the domain.
- Feedback from the collection agents in the domain.
- Information from collection coordinators in other domains.
- Feedback information from the collection monitor in the domain.

Only the first type of input, alert messages from collection agents, is currently used by the system. The eventual use of the other types of input is deferred for future work.

Information from a collection coordinator in one domain to a collection coordinator in another domain can be valuable if the collection coordinator in the first domain detects that an attack is spreading to the domain of the second collection coordinator. The other tree types of input are all feedback information from components in the collection coordinators domain that may improve the accuracy of the collection coordinators orders.

Alerts from intrusion detection systems differ from the four other types of input in that it is generated from an entity that is not a part of the system. The format of the alert is determined by the IDS that generated it and different alert format may contain different information. The design and implementation of the collection coordinator is simplified radically if alerts are received in a standard format, since then the collection coordinator only needs to support that alert format.

Snort is probably the most widespread NIDS today. It is developed as an open source project. Both the source code and the signatures are available for review. The format of alerts that Snort [13] generates contains a snort-id that uniquely identifies the alert. The snort-id is as it name suggests only supported by Snort. The alert also contains a category, a priority and possibly one or many references to vulnerability databases or dictionaries, like CVE [50] or Bugtraq [51]. The name of the alert often includes the vulnerability that is exploited and/or the attack tool being used.

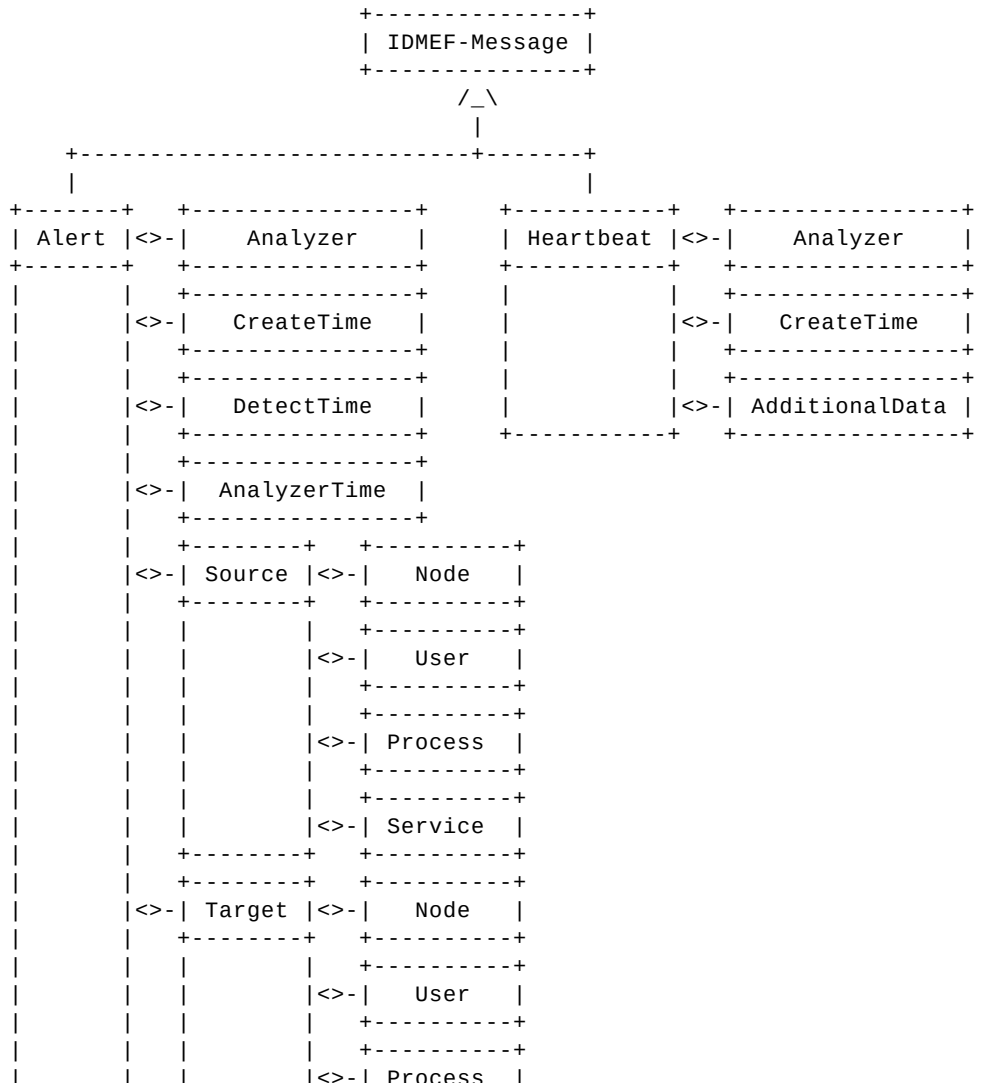

Figure 6 IDMEF data format

The fact that individual IDS use proprietary formats for alerts and that many organisations use many different IDS, has lead to the need for a standard alert formats, so that alerts from

different IDS can be correlated. The Intrusion Detection Message Exchange Format (IDMEF) [26] is a proposal of such a standard format put forward by the Intrusion Detection Exchange Format working group (IDWG).

IDMEF presently defines two types of messages: Alert and Heartbeat. The data model of IDMEF is object-oriented. The Alert message type, which is of most interest in relation to the functionality of the collection coordinator, contains a number of different classes, each with its own attributes and aggregate classes. The Classification class and the Assessment class are more interesting then others from the collection coordinators point of view:

The Classification Class contains two classes. The name class that contains the name of the attack and the URL class which contains an URL at which one can find a description of the alert. The Classification class also contains an attribute called origin. The permitted values of origin are currently: unknown, bugtraqid, cve and vendor-specific.

Common Vulnerabilities and Exposures (CVE) is a list of standardized names for vulnerabilities and other information security exposures. CVE aims to standardize the names for all publicly known vulnerabilities and security exposures. When a new vulnerability is discovered it is assigned a candidate number. After review the candidate may be assigned a CVE name. Bugtraq is mailing list for discussion and announcements of security vulnerabilities and a vulnerability database.

Since not all alert are vulnerabilities and CVE and Bugtraq classifies vulnerabilities, the cve and the bugtraqid don't provide a complete and consistent alert taxonomy [48]. That is not all alerts contain a Classification name from cve or bugtraqid.

The Assessment class aggregates by others the Impact class that contains three attributes, severity, completion and type. The severity attribute holds an estimation of the relative severity of the event. The completion attribute gives an indication whether the IDS system that sent the Alert believes the attempt was successful or not. And lastly, the type attribute contains a classification of the attack.

admin	Administrative privileges were attempted or obtained
dos	A denial of service was attempted or completed
recon	A reconnaissance probe was attempted or completed
user	User privileges were attempted or completed
other	Anything not in one of the above categories

Table 4 Table of IDMEF classifications

IDMEF defines three subclasses to the Alert message class: the ToolAlert class, the OverFlowAlert class and the CorrelationAlert class.

The ToolAlert class carries additional information related to the use of attack tools or malevolent programs such as trojan horses. With the ToolAlert class one or more previously sent alerts can be grouped together, to indicate that the alerts were all the result of someone using a particular tool. The ToolAlert class contains information about the name of the tool and the commands used in executing the tool.

The OverFlowAlert class carries information related to buffer overflow attacks. It contains the name of program the overflow attack attempted to run, e.g. a shell, the size in bytes of the overflow and the overflow data itself.

The Correlation class can be used to group one or more previously sent alerts together, to indicate that that they are related. The class contains a string describing the reason for the grouping, for example a particular correlation method.

The IDMEF Alert message contains other information that is of essence to the collection coordinator. Especially the Target class is of importance. The Target class contains information that identifies the node (network address, network name etc.) being attacked, and possibly the user id used in the attack, the process that is being attacked, the network service being attacked, and files on the target involved in the attack.

6.4.1.2 Generating collection orders

The process of generating a collection order consists of two activities:

- Identify which nodes to send collection orders to. This can be the host(s) under attack, but also other nodes like the firewalls, routers etc.
- Generate the content of each collection order, the information about what collection to start.

Note that an alert message can result in a number of unique collection orders addressed to a number of different destinations. Which nodes to send collection orders to, the first point above, is treated in the next section. The format of the collection orders is treated in section 6.4.1.4.

The content of the collection orders can be decided in a number of ways:

- Use a proprietary identifier, like the snort-id. The advantage is that the translation from alert to collection order becomes straightforward.
- Use a standardized identifier, like the CVE name or bugtraqid.
- Use the alert's classification or category, like the IDMEF impact type or the snort Classtype rule option.
- Use information about the attack tool used, supplied by IDMEF's ToolAlert class. Attack tools can be analysed to determine which data that needs to be collected to capture their actions.
- Use information about the program being attacked, the user id used in the attack, the process that is being attacked, the network service being attacked, and files on the target involved in the attack, supplied by the two IDMF classes OverFlowAlert and Target.

A combination of the listed sources of information above can of course be used. The information that different alerts contain varies so different information sources must be used. However, it can be assumed that some form of id (either proprietary or standardized) is always included in the alert.

The mapping functionality, from input to collection order, is implemented by a number of tables or functions, one for each type of input. In this way the translation functionality is easily extendable. If a new input type is to be supported a new table or function is added and the old are left unchanged.

When an alert message is received its content is matched against the available tables and functions. The tables and functions which input is present in the alert return as result a set of collection orders. A table or function may return many collection orders but each order has a separate destination address from the others. Then, the resulting set of collection orders is grouped after destination. Finally, each group of collection orders is merged to form the collection order to each destination. Collection order format and merging is discussed in section 6.4.1.4.

The tables can be of varying complexity or size. In the simplest case the table only consist of one entry that specifies a standard set of collection orders. In this case all detected attacks results in the same set of collection orders.

The specification of the semantics and the syntax of the tables and functions are deferred for future work. The tables and functions must evolve over time, as the signatures of IDS systems do. However, how the mapping functionality is maintained is still an open question.

The mapping functionality is currently defined as state-less, the mapping is only determined by the current input and not by past input. State-full mapping would make it possible to construct collection orders based on a sequence of alerts.

6.4.1.3 Where to send collection orders

The identification of which nodes to send a collection order to is decided by the same tables and functions that decide the content of the collection orders. Usually the collection order generation results in collection orders to the host or hosts under attack and sometimes also to server and network component nodes.

The destination may be a symbolic name such as firewalls, routers etc or an absolute name, typically the hostname or IP-address of the attacked host. The symbolic names are resolved in the last step of the collection order generation, possibly resulting in that the collection order is sent to several nodes.

The symbolic names are mapped to network addresses (IP-addresses) in the configuration file of the collection coordinator.

Except the host(s) under attack there are numerous nodes that the collection coordinator could send collection orders.

The currently defined symbolic names are:

- network_sniffers
- routers
- firewalls
- fileservers
- directory_services
- authentication_servers
- email_servers
- modems_pools
- vpn_servers
- web_servers

A copy of all collection orders is always sent to the log server.

6.4.1.4 The format of collection orders

Each collection order has one destination and the collection orders contain information about the type of collection that the local collection agent shall start if possible. The collection agent will examine the collection order and follow it according to its collection capabilities.

How precise should the collection orders be? Is it enough to define log levels and define the collection to take place in the different levels? Or should the orders be more specific? They could contain a collection set of adapters types and the collection that each adapter type should start.

Since the results from the functions and table lookups should be merged, the format must include a definition of the merge operation. For log levels the natural definition is to let the result be the highest log level among the merged. For collection sets the natural definition of the merge function would be the union function.

The format of the collection orders must be further evaluated. The prototype implementation presented in chapter 7 uses log levels.

6.4.1.5 When to halt collection

Below is a list of different methods for determining when the collection should be halted:

- Manually by an operator. Only feasible if the number of alerts is low. Otherwise too much of the operator's time will go to determine when to halt collection.

- Use pre-defined time constants, possibly infinite, for different attacks. That is, after x seconds is the fine-grained logging turned off, where x is dependent on the attack. If x is infinite the operator must stop the collection manually.

- The log analyser could analyse the collected data to determine in real-time when it's safe to turn of the collection. The analysis could for example be based on system properties like: users logged on; open network connections and the processes

running on the node in question. If these system properties indicate that the attacker has left the node and left no running processes behind him the collection could be stopped. The analysis will also indicate if the attack was successful or not. If so, the node should be manually examined after the collection has stopped.

It's currently not defined how collection is halted. To use predefined time constant is easy to implement and offer better functionality then halting the collection manually. In the long run the method based on analysis of the collected data is preferable. However, to get there more research needs to be conducted.

6.4.1.6 Configuration

The configuration of the collection coordinator consists of three parts: the mapping tables and functions, system integrity configuration, and the communication security configuration.

The integrity configuration consists of the key material used to sign all messages sent by the collection coordinator to assure integrity and trace ability.

The collection coordinator's communication security configuration specifies:

- The identity and credentials of all the other components except the log in the domain
- The identity and credentials of the collection coordinators in other domains that the collection coordinator should cooperate with. It also contains the range of network addresses that is included in each of these domains

6.4.2 Collection agent

Figure 7 gives an overview of the system architecture on a collection node. The targets can be local processes that generate data that the system collects but the adapter can also communicate directly with some part of the operation system. The collected data can be log-data from for example a web-server, alerts from an IDS, output that is generated by running a system command, or statistics about network load, etc.

The collection agent collects this data and is responsible for the communication between the local system and the log server, the collection coordinator and the collection monitor. The collection agent communicates with the targets through the adapters. Usually the adapters collect output from their targets and send it to the collection agent, but the adapters may also collect system data on their own (directly from the OS). The adapters control what kind of data that is collected, and the format of the data.

To minimize the delay from attack to collection starts is it important that alert messages is relayed to the collection coordinator and that collection orders are processed as fast as possible.

The local collection system has an extendible design, which is implemented through adapters and plug-ins. Plug-ins are built into to the collection agent at compilation. It should be easy and require a minimum of knowledge about the system to make a new adapter. The interface that adapters need to implement should be clearly defined. It is also desirable that a

new adapter can be added without any change to the collection agent, except a change of its configuration to include the new adapter. A version of the adapter interface for Linux is defined in Appendix A.

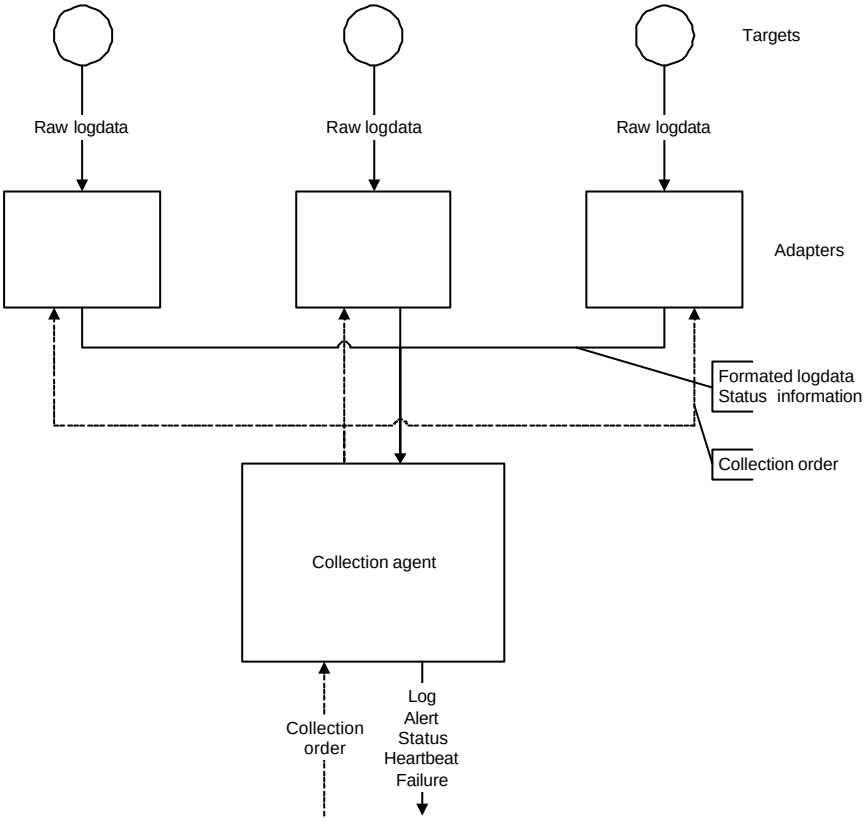

Figure 7: The design of the system on a supervised machine.

The collection agent should avoid starting new processes, opening sockets etc when fine-grained collection is started. The collection agent should be as silent as possible, to not arise suspicion from an eventual attacker.

6.4.2.1 Architecture of the collection agent:

The collection agent controls its adapters. It starts, stops and reconfigures its adapters. The collection agent collects all the messages from the different adapters, and sends them to the log server. It receives collection orders from the collection coordinator and reconfigures its adapters according to them. Finally, the collection agent monitors the targets through the adapters and is responsible for forwarding this information through status and heartbeat messages.

The agent may be extended to implement functionality to process the streams of data sent from the adapters. Examples of processing are filtering, correlation of data, or implementation of intrusion detection or analysis capabilities. The functionality is added through the plug-in interface. There is one plug-in chain for each of the log, alert, and status message streams from the adapters. When a new message is received from an adapter it is handed over to the corresponding chain. The message then travels trough each of the plug-ins in the chain, before it is processed by the collection agent and finally forwarded. Note

that it's possible to add plug-ins to the alert chain but it should be carefully contemplated so that no unnecessary delay is introduced.

The plug-ins compiled with the collection agent must be documented. Every type of plug-in is assigned a globally unique id. The id consists of the name of the author, and the name and version of the plug-in. The ids of all plug-ins that a collection agent uses are included in the status message sent at start-up.

Figure 8 illustrates the simplified workflow of the collection agent. The figure does not include the sending of heartbeat and the processing and sending of status messages.

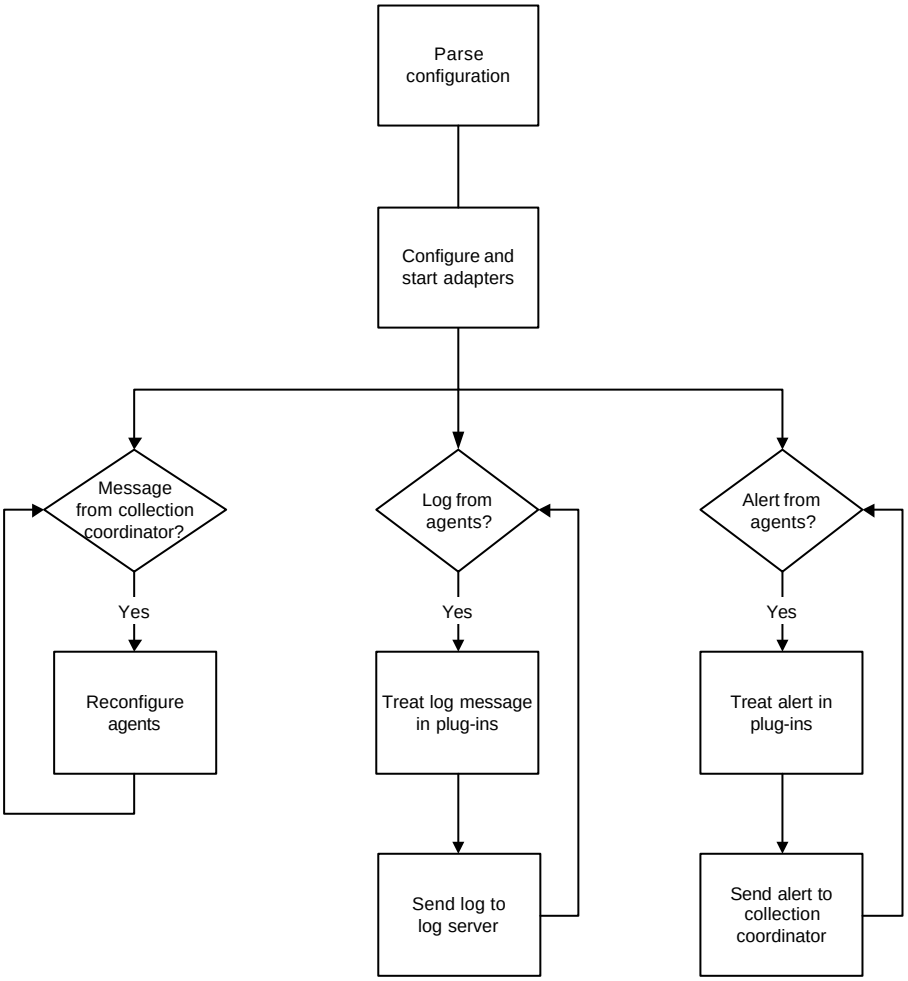

Figure 8: Workflow for the collection agent.

The collection agent starts up, and reads the configuration file. Then it starts all the adapters according to the configuration, and starts to listen for messages from the collection coordinator, and for messages from the adapters.

6.4.2.2 Adapters

The adapters provide the system with an interface to the target. That means that each adapter is specially constructed for a single target, or a group of targets with the same functionality. The adapters collect the data that the target generates. Thereafter it has to

convert the message into a format that the other modules of the system understand, before sending it to the collection agent as either a log message or an alert message. The adapter attaches a tag to the message, describing the contents of the message (format id) and the time interval during which the data was collected.

The adapter also generates status messages when its status changes. Status change events include: start-up and shutdown of the adapter, and when it receives new orders about what to collect. The adapter should also monitor its target, and generate a status message if the status of the target changes, for example if it goes down.

If the target is a process (application or a process that collects system information) the adapter can check the liveliness of the target process through e.g. periodically polling the target if possible, listen to eventual periodic output from the process etc. If the target offers hooks to connect to, for example SNMP trap, the adapter may listen to these. The target process can potentially be monitored in many different ways. It needs to be examined in each case which of the possible ways that is preferable.

The target must not be a process. It can be a part of the operating system, for example on Linux systems there is interesting information to be read in the /proc directory. In that case the adapter itself implement the collection functionality. This is clearly preferable since the adapter then doesn't have to trust and monitor any target process. The adapter must however trust the operating system. On for example Linux it's not obvious that the OS can be trusted since it can be modified with loadable kernel modules.

If it's possible to reconfigure the logging of the target process, the adapter should be able to receive collection orders from the collection coordinator via the collection agent, and reconfigure the logging of the target process according to these. If the target process does not offer the possibility to be reconfigured the adapter has to filter the data from the target according to the collection orders.

Every adapter is assigned a globally unique type id, which is made up the name of the author, and the name and version of the adapter. The collection agent includes the unique type ids of all the adapters it uses in the status message sent at start-up (included in the configuration of the collection agent) and when the adapters are reconfigured as an effect of a collection order.

It is desirable that the adapters are executed within the process that executes the collection agent. The adapters in the prototype implementation (see section 7.5) are executed in the same process as the collection agent. Otherwise the collection agent must somehow monitor its adapters. If the adapters execute in the same process as the collection agent the collection monitor also monitors them.

6.4.2.3 Communication between adapters and targets processes

There are several constructs for inter process communication that can be used by an adapter to communicate with a target process, including:

- Pipe (e.g. target | adapter)
- Through the target's log file
- Sockets
- Named pipe
- Plug-ins in the target
- API supported logging

Not all construct are available in all operating systems. Pipe and communication through the targets log file offer only unidirectional communication. The other types may offer bi-directional communication.

To ensure integrity, it's feasible to enforce some authentication mechanism between the target and the adapter if possible. However, it is of course only possible if the target supports it.

If the communication uses constructs like sockets, named pipes or files the access rights to theses constructs must be set so that the attacker must gain super user privileges to access them. An attacker with access to one of these constructs can send false information through them or disrupt the communication.

If the target supports some sort of plug-ins mechanism or API that enables transfer of information between target and collection agent it is preferable over the other communication alternatives. There has not been conducted any in-depth investigation so far in the project over currently available plug-ins and APIs for exchange of logging information. In section 8.3.4, covering possible future work, the desirable functionalities of an application plug-in or API for transfer of application logs and monitoring is discussed further.

6.4.2.4 Configuration

The adapters are configured in the collection agent's configuration file.

The key material used to sign the log messages (the collected data) and other messages sent by the collection agent must be configurable.

The collection agents must be configured with the identity and communication security credentials of the collection coordinator, collection monitor and log server in the domain.

6.4.3 Log server

The function of the log server is simply to receive the logs from all the other components in the system, and store them. The log server should log all messages generated by the system, that is all log messages, alert messages, collection orders, status messages and failure messages. Implying that the log server is both responsible of storing the data collected by the system and the data that describes the operation of the system.

Figure 9 shows a proposed design of the log server, where the log server is connected with the production network via a one-way connection, while all access to the data in the log server is done via a dedicated administrative network. A one-way connection can be

implemented with firewall rules or for higher security by cutting the outgoing connection physically if the network medium uses different cables, for incoming and outgoing traffic.

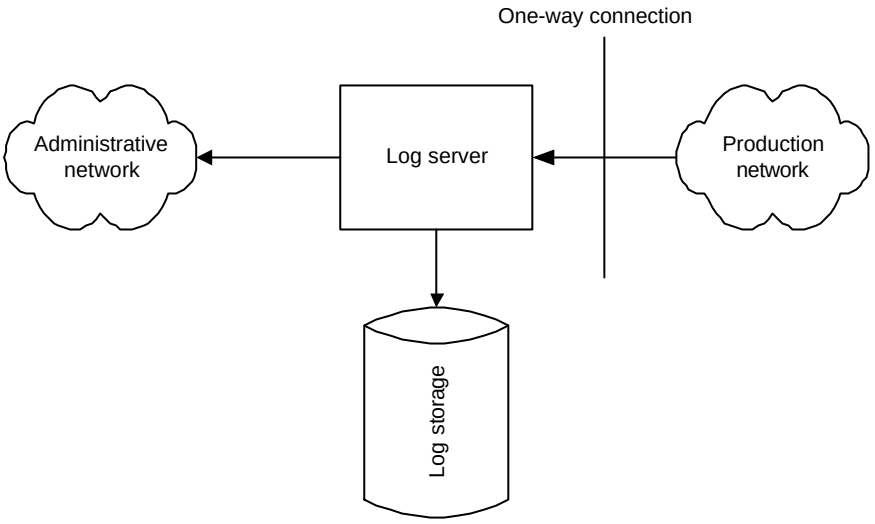

Figure 9: The design of the log server

6.4.3.1 Storage

The data in the log server can be stored in many different formats, for examples in text-files, as XML-documents, or in a database. The log server is designed so that the storage is handled by a plug-in, compiled into the log server. This way it's easy to extend the log server to store in a new format. For most formats it's necessary for the log server to parse the log messages that arrive. This makes analysis easier, but the original form the messages arrived in is lost. A solution is to both save the data in the received original format and the common format.

The data should be stored in such a way that relevant data can be efficiently found. For example should it be easy to search after all logged data for a particular host and a particular period of time or all data collected as the effect of a specific collection order.

6.4.3.2 Security

The confidentiality and integrity of the collected data must be protected. No access to the stored data is allowed through the management network. Authorized traffic coming from one of the other components in the domain is only allowed to append information to the stored information. The data can only be accessed through the administrative network, and then only by the domain's log analyser and management console. Stored data cannot be deleted without physical access.

These access constraints are implemented by a combination of firewall rules and file permissions. In addition, the log server should be located on a dedicated host with an operating system only supporting a minimum of functionality.

The stored data should also be encrypted and signed with multiple hash algorithms, as stated in the requirements section.

For a higher level of integrity protection, the signatures (signed hashes) could additionally be saved on a separate node. This feature would be motivated by the fact that an attacker that gains access to the log server may modify the saved log records and corresponding hashes. If the hashes are only saved locally on the log server this unauthorized alteration will not be detected. The design does not currently include this feature. Since the collected data and all messages are already signed by the component that collected the data or sent the message there is already some protection. To falsify the data stored in the log server an attacker must acquire the key used to originally sign the collected data or message and acquire access to the log server.

In addition, as mentioned above, may the log server be physically restricted to send over the production network if the network medium allows it.

6.4.3.3 Configuration

The log server must be configured with the identity and communication security credentials of all other components in the domain.

The log server must also be configured with the key material used to sign the received data and messages.

6.4.4 Collection monitor

The collection monitor is responsible for monitoring the other components of the system. It compiles information about the system's state from the status, heartbeat and failure messages it receives from the other components. The system's state can be monitored through the management console and the collection monitor can be configured to alert an operator through mail, SMS etc.

The functionality of the collection monitor is described in section 6.2.1, 'Component monitoring and integrity'.

6.4.4.1 Configuration

The collection monitor must be configured with the identity and communication security credentials of all other components in the domain.

The log server must also be configured with the key material used to sign the received data and messages.

7 Implementation

As a part of the project a prototype has been implemented. The prototype implements parts of the design proposed in section 6. We didn't have time to implement all the components, or all the functionality, so we focused on implementing the core functionality, remote logging and logging based on detected attacks. All requirements have not been implemented, but the prototype is designed to simplify the implementation of this functionality later.

In the first phase of the implementation, we focused on making an extensible architecture that is capable of fine-grained logging. We especially focused on that it should be easy to add new adapters. Therefore it was the interface and interaction between the collection agent and the adapters that got the most attention.

The user documentation for the prototype is given in Appendix A, where the installation, and configuration instructions for the prototype are given.

We have implemented prototypes of the log server, collection agent, and the collection coordinator. We've also implemented a number of adapters: one for collecting alerts from Snort [13], an adapter for collecting system-calls, a generic adapter that listens to a named pipe and collects everything written to that pipe, and a adapter that runs system-commands and collect the output of these commands.

The prototype is implemented on Linux and written in C. The reason for using Linux is that it's a standard Unix-like operating system, which has a large market share of the server market. We use C since it's a much-used programming language that offers good low-level interaction with the operating system.

7.1 Communication

7.1.1 Communication protocol

Since the requirements state that the communication between the components of the system should be reliable it must be based on TCP. Both HTTP and, IDXP or BEEP, are all application protocols built over TCP that offer additional services compared to TCP.

IDXP [25] stands for intrusion detection exchange protocol, and is a protocol that is specified by the Intrusion Detection Workgroup (IDWG) at IETF. It's meant to be a standard for transactions of intrusion detection data, especially in conjunction with IDMEF [26]. IDXP uses BEEP, a generic protocol over TCP where characteristics like confidentiality, integrity, etc. can be added using profiles. For an introduction to BEEP see [28].

The advantage with BEEP is that it's a generic toolkit for constructing application protocols where many communication mechanisms, like framing, encoding, negotiating capabilities, already are implemented. BEEP lets the application protocol designer choose with mechanisms to include through the construction of profiles. Another advantage with IDXP is that if IDXP becomes a standard, it'll be easier to incorporate our components with other

components that talk IDXP. A problem using BEEP is that it's a new technology, so it hasn't matured yet. It's also fairly complex to develop your own profiles. Codefactory are working on an implementation of IDXP, but the implementation isn't stable yet. See [23] for details about the implementation.

Developing our own protocol from scratch using sockets could give us the most efficient protocol, since the protocol could be custom made, it's possible avoid some of the overhead in a more generic protocol like BEEP, or HTTP. It's possible to implement security using VPN, or SSL [29]. The problem with this approach is that it's a considerable amount of work to implement everything from scratch, protocol design is hard and it's easy to make mistakes, and it's troublesome to implement security features like confidentiality, integrity, etc.

If we use HTTP, we can reuse all the services that HTTP offers. Implementing security is possible, and can be implemented in a standard way using SSL (HTTPS).

We decided to use IDXP, because it seems like it'll become a standard, and it's relatively simple to extend the protocol to fulfil the system's security requirements. But since the implementation of IDXP still wasn't mature at the time when we started implementing our prototype, we where forced to construct a simple application protocol, while we wait on IDXP to mature. This application protocol is built on top of standard TCP-sockets. However, one should note that the prototype is prepared for the use of IDXP. As an effect of not using IDXP initially the security requirements on the communication is not fulfilled in the prototype.

7.1.2 Message formats

Three of the message types that are presented in section 6.3.2 have been implemented, the log, alert and collection order messages.

The prototype implements a simplified version of the message header defined in section 6.3.2. Only the Type, Format, and Payload length fields are included.

The only implemented format for log messages is the raw format. In this format the log messages contain log-records sent as a text-strings without any additional metadata.

Alerts can be sent in two different formats. They can either be sent as a text-string, or they can be sent as a data structure. If they're sent as a text-string, the receiver of the alert has to parse the string to obtain the information needed to process the alert. The data structure format presently contains the alert message, the attacked host and the attacked port.

Collection order messages are implemented using the log-level approach as described in section 6.4.1.4. The log-level should be interpreted as an indicator on how serious the attack is.

7.2 Collection coordinator

The implemented collection coordinator is a simplified version. It generates the same collection order for all alerts received. It listens for alert messages from collection agents. If it receives an alert message from a collection agent with an IDS target, it will send a collection order to all the connected collection agents, independent of the contents of the alert. In the prototype the log level is set to a constant value, independent of the alert that the collection coordinator receives.

The collection coordinator is configured with the listening ports of the different components, and the addresses of the log server, and the collection agents.

7.3 Log server

The prototype log server implements a minimum of functionality. The only thing it does is that it receives log messages from the collection agents, and stores these in a normal text-file. The name of the text-file and the listening port are configurable.

7.4 Collection agent

The collection agent defines an interface for the adapters to implement. The adapters are implemented as libraries that are loaded dynamically at run-time. The adapters are initialised with a pointer to a callback function that is used to send messages back to the collection agent. The adapters init function returns a structure that contains information about the adapter, among other things functions to call when you start the adapter, when you stop it, and when you change the configuration. The messages from the adapters are put in a queue according to type: alert, log or status. These messages in the queues may be processed (filtered, extraction of statistics etc) before they are finally sent to the log server.

The collection agent also listens for messages from the collection coordinator, and if it receives a collection order, it instructs its adapters to start extra logging.

Listening ports of the collection agent, log server and collection coordinator are configurable. The same goes for the addresses of the log server and the collection coordinator, and the paths of the adapter's shared object-files.

7.5 Adapters

As explained earlier are all the adapters implemented as libraries that are loaded dynamically at run-time. This way more adapters can be added to the collection agent, without recompiling the agent. To add an adapter it's enough to add the location of the compiled shared object-file for the adapter to the configuration file of the collection agent and for the new adapter to be loaded the collection agent only needs to be restarted. All adapters implement a common interface that defines functions that the adapter must implement. For details on this interface see A.5.

The adapters are configured in the same configuration file as their collection agent.

So far we have implemented four adapters, the pipe adapter, the run adapter, the snort adapter and the auditd-adapter.

7.5.1 Pipe adapter

The pipe adapter is a general adapter that listens to a named pipe. All messages that the adapter receives on the pipe are sent as log messages to the collection agent. The name of the pipe is at the moment hard-coded, but it should be configured in the collection agents configuration file.

7.5.2 Snort adapter

This is an adapter special for the Snort IDS. It receives alert structures from Snort through a UNIX socket, and transforms the alerts to a standard alert format. The alerts are then relayed to the collection agent.

7.5.3 Run adapter

This is a simple adapter that runs a UNIX system command, and sends the output from this command to the collection agent. The command that is run is at the moment hard-coded, but it'll be configurable from the agents configuration file later.

7.5.4 Auditd

The kernel is a protected area of the operating system, which is only available through system calls. System calls are the connection between kernel space and user space, where user space is the logical area of memory and CPU usage where user processes run. All activities on a computer, e.g. open a document or start a program, results in system calls. Hence, the obvious place to be sure all user activities are logged is the system calls.

When a system call is executed, the PID of the process calling the system call can be identified. From the PID the real user ID of the user executing the system call can be identified. This makes it possible to log useful information about what actions a user perform and when they happen. The following is an example system call log entry from a package called SNARE [12]:

dhcp62.nr.no LinuxAudit event,open(O_RDONLY),Thu Jul 11 16:16:05 2002 user,root(0),root(0),root(0),root(0) process,24724,snare path,/proc/auditinfo return,8 sequence,75098

As can be seen, the machine name, system call, date, time, real user ID and real group ID are presented, as well as other useful information.

SNARE is developed by InterSect Alliance, and released under GPL. They use a well-known principle for recording system calls, by using a kernel module. A kernel module can be inserted into the kernel at run-time. This makes it possible to install such a module without recompiling the whole kernel or rebooting. As this module is run in kernel space, it can attach itself to and be called via the proc file system. The kernel module can then access internal features of the kernel, giving user space access to the kernel via the inserted kernel module.

The most interesting capability in our setting is modifying the system call table, which gives references to where handlers for the different system calls can be found. By using a kernel

module, we can access and replace the entries in the table with our own, located in the kernel module we inserted. From our kernel module we can then call the original system calls. We have now intercepted the system calls, and can find the information as given in the example above by asking for the current user PID and time, as well as record what kind of system call the user requested. The result may be a string as the one in our example above.

In our solution, we have modified the user space part of the SNARE package to run as an adapter. The kernel module is kept as originally written. The basic steps performed to be able to monitor the use of system calls are these:

1. Insert the kernel module.
2. The kernel module registers itself when inserted, adding a file called auditmodule to the proc file system.
3. A (user space) process calls the kernel module via functions registered in the proc file system to start logging of specific system calls.
4. The kernel module registers the calling process' PID.
5. When a system call is intercepted, the kernel module logs it in a linked list, and notifies the calling process that requested the logging.
6. The process requesting logging ask for the next structure in the linked list held by the kernel module to be returned via the proc file system

7.6 Portability

The main system with the agent, coordinator and the log server should be easy to port among different UNIX-platforms. The same goes for the Pipe adapter, snort adapter and run adapter. When it comes to porting to a non-POSIX platform, for example Windows, is becomes much more difficult. The main reason for this is that the system is implemented using POSIX threads. When it comes to the auditd adapter it can only be run on Linux or Solaris, since the audit module only is implemented for these platforms.

It may be necessary to be able to catch signals in the adapters. The auditd-adapter already communicates with the kernel module using signals. This can give problems on traditional UNIX platforms, since it's not possible for another process to easy send a signal to a specific thread. On Linux this presents no problem however, since in Linux every thread is assigned its own processor id, and therefore has its own signal handling.

8 Future work/Research directions

This chapter presents a number of areas and issues related to the work presented in this report that needs further work. The work needed to complete the system is presented, including the prototype and the issues that need to by be examined and clarified in the design presented in section 6. Functionality and ideas regarding the design not presented before are also presented. Lastly are a number of related areas of future work and research presented, that either have been identified during the course of the project or that have not been addressed so far in the project (not prioritised in this phase of the project).

8.1 Evaluation

So far in the project the hypothesis and the system designed based on it has not been practically evaluated. For an evaluation to be of any value the functionality of the prototype needs to be extended to include monitoring, a more advanced implementation of the collection coordinator, authentication etc. A more complete prototype can be used to validate and test the hypothesis and the design against real attack scenarios. In the first phase the aim should be to validate the core idea of the hypothesis that is starting collection based on alerts from intrusion detection systems. Thereafter can the design be evaluated against the requirements put on the system.

Tests need to be conducted to quantify the delay from the time of the attack until collection is started. The components of the delay (delay caused by processing in IDS, collection agent, and collection coordinator and delays caused by the communication) should also be quantified.

Test is also needed to get a sense of the resource consumption that collection of different types of data, like system calls, introduces.

8.2 The design

This section presents a summary of the issues that needs to by be examined and clarified in the design presented in chapter 6 and some ideas of new functionality that have not been presented before.

Fault tolerance (traditionally implemented through redundancy) is currently not addressed by the design. However, the design is aimed at simplifying introduction of fault tolerance through: simple design, components with clearly defined responsibilities, and state-less components.

The syntax and semantics of the tables and functions that implement the mapping from alerts to collection orders needs to be specified. Different models for the maintenance and development of these tables and functions will also in the long run be necessary to investigate.

The format of the collection orders also needs further work. In section 6.4.1.4 two ideas for formats are discussed, log levels and collection sets.

How to determine when to halt fine-grained collection was discussed in section 6.4.1.5. A number of different directions to further investigate were presented. The most advanced method, to analyse the success and progress of the attack, requires work on adding analysis capabilities to the system. The design identifies a log analyser component for this purpose, but analysis can also be conducted in other components, or in new components that are not currently part of the system.

In section 6.4.1.1 a number of different input sources to the collection coordinator were presented. The only input currently used by the system is alert messages from the collection agents in the domain. The need for and the eventual mechanisms used for inter-domain communication between collection coordinators needs to be further investigated. The same goes for the intra-domain feedback from collection agents, log analyser and collection monitor.

The design identifies two components that are not specified in any detail, the log analyser and the management console. The responsibilities and functionality of these components needs to be specified.

How the collection monitor handles heartbeat messages needs to be further specified. Especially how the collection monitor calculates the time of the normal period between two heartbeat messages. The design currently only suggests that the calculation can be based on a configured period and statistics on normal deviations from that period.

The design states that the integrity of the system executables should be protected by automatic and manual integrity checking (see section 6.2.1). It is further proposed that the documentation of the results of the integrity checks should be transferred to the log server. The mechanism that transfers the documentation needs to be designed.

The keys used to sign the data must be managed. The public keys corresponding to the private keys used to create signatures must be stored so that they can be accessed during analysis and their integrity must be protected, while the secrecy of the private keys must be protected. It needs to be further investigated how the signature keys should be managed (i.e. updated, stored and secured) and which cryptographic algorithms that should be used.

In section 6.2.3 a number of countermeasures against denial of service attacks are listed. The feasibility of the identified countermeasures and where in the system they should be implemented need to be analysed.

Which events the individual components should log and how these logs are transferred to the log server are currently not specified by the design.

The design does not treat how the integrity of the communication that carries the time synchronisation information between timeservers and components is protected.

It is not currently defined if symmetric or asymmetric cryptography is to be used to ensure communication security. The advantages and disadvantages of the alternatives are discussed in section 6.3.3.

It needs to be analysed how different types of data collection taint the system. The system must include the possibility to specify the order of collection so that collection that taints the system is conducted after the tainted data in question is collected. This implies that the tainting of the different targets used by the system needs to be identified.

How the collection of different types of data is affected by the delay described in section 3.2 needs to be investigated. The use of a small local buffer to circumvent the problem also needs to be investigated.

Currently the analysis of the collection coordinator is state-less. The need for and the possibilities with state-full analysis need to be further analysed. State-less analysis simplifies the design of the system but the introduction of state makes new analysis functionality possible. As an example IP addresses that have been used in former attacks can trigger fine-grained collection if they reappear again within a certain interval. More complicated analysis like the prediction of the attackers future behaviour also requires the introduction of state.

By predicting the attackers behaviour the system could pro-actively start collection. In [49] a profile of the attacker is created based on information he/she reveals about himself/herself during the attack. Principles from economics are used to predict the attacker's behaviour, based on estimates of his/her asset-appraisal, attack-cost, and attack-resources. Based on the profile and economic-based estimates likely compromised devices are identified. In [47] analysis of the attacker's attack strategy is proposed to be able to perform "pro-active look ahead adaptive auditing".

The collection agent could be put in kernel space. In the current design it resides in user space. The main advantage of running the collection coordinator in user space is that it is easier to port it to different platforms. If the collection coordinator is placed in kernel space, for example as a loadable kernel module in Linux, many of the monitoring and integrity requirements put on the system are easier to fulfil, since the collected data then can be transferred directly to the other components without passing through user space.

8.3 Related research directions

8.3.1 What to log

We believe that forensic collection on a live system (i.e. logging) is a field within computer security that has not yet got the attention it deserves. The literature study conducted in the project has uncovered only a few papers in this field. Further work is needed in classifying different types of log data, analysing the relation between the classes and their value from different perspectives, as for example their value in detecting intrusions or making a profile of the attacker.

8.3.2 Report format

One of the goals of the hypothesis that this project is based upon is to simplify the collection of computer forensic evidence. To simplify the part of the forensic process after collection a standard format for reporting computer intrusions is needed. Tool development would be greatly simplified if a standard format existed. Collection tools could deliver the data in this format and analysis and presentation tools could work against this format. The work of the human investigator would also be simplified if all incidents where reported in the same format.

8.3.3 Enforcement of security properties

As discussed in section 1.1.2 the system presented in this report can be used to collect data to enforce security properties. The project has not addressed this application of the system so far. To automatically detect events that breach a local security policy, it must be specified in a formal language so that it is possible to automatically derive the set of events that breaches it. To specify a whole security policy in a formal language is not feasible today, but parts of the security policy can be specified this way. Parts of local security policies are already today specified in formal languages. As examples firewall and access control configurations can be mentioned. Based on these formal specifications it can be determined which events that are of interest and that therefore should be collected.

8.3.4 Standard for logging support in applications

The fine-grained approach would benefit if applications offered standard APIs for controlling their logging. Currently the logging (what to log and where to store the logs) of applications is often controlled through configuration files and the logging is modified through modification of these configuration files and the restart of the application (or possible only the logging process of the application). Modification of configuration files and restart of the application is a slow process and logging is not activated during the restart.

Monitoring functionality may be included in the API. If so, the monitoring process would use the API to subscribe to information about the application's state changes.

All commands received through the API should be logged, so that information about all changes to the logging is distributed to all interested. This is to protect the integrity of the level of the logging conducted by the application. Otherwise, a hacker can change the level of the logging to not include events that would reveal him/her without being noticed.

To our knowledge no standard or proposed standard API for controlling logging exists today.

8.3.5 Standard logging module in operating systems

The system presented in this report could be extended with a logging module that logs the users' interactions with the operating system. A standard API controls the logging of the module, much like the logging API for applications discussed above.

The logging module should be logically separated from the rest of the system (its actions should not modify the system state), that is for example its actions should not modify MAC

times of files it accesses. The module should also be "impossible" to remove (compiled into the kernel). Only the level of the logging and where the logs are stored or sent can be modified and the modifications are logged.

A logging module prototype for Linux can be developed in the form of a loadable kernel module. A loadable kernel module can be removed by unloading it, but the module can log the unloading event before it is unloaded, so the unloading is detected. The logging module can be integrated with the system prototype, through an adapter for the logging module. The logging module should however be instructed to send the logs directly to the log server, and not pass the logs to the collection agent. The collection agent resides in user land and if the logs are passed through the collection agent it makes it possibly available for a hacker with super-user privileges.

Further work is needed in what logging capabilities the logging module should have. The project has so far identified a number of candidates (see section 4), as for example system calls, file access, loading of kernel modules etc.

8.3.6 Reduction of false positives

The data that the system collects can be analysed to rule out IDS alerts as false positives. This resembles the functionality of ClearResponse from Psionic. The work started in collecting system calls could offer a base for further analysis. If the pattern of system calls executed when exploiting different vulnerabilities could be identified, the success of attacks taking advantage of these vulnerabilities could be measured.

8.3.7 Isolation of attacked nodes

The system presented in this report passively collects data during presumed attacks. It would clearly be desirable to use this collected data in real time to determine if attacks are successful and if so isolate the attacked node from the attacker. Resulting in either terminating the attack or hindering the attacker from returning and use his/her acquired privileges. In any case the harm the attacker has caused is limited and the examination of the attacked node can be conducted without risking that the attacker returns. In the cases the attack is determined successful it is possible to redirect the attacker to a deception system, a so-called honeypot or honeynet. However, to determine if an attack is successful or not is hard since the accuracy must be high, since one cannot accept isolation of nodes that in reality are not compromised (or at least not very often).

8.3.8 Analysis of the collected data

The project presented here has not addressed issues related to the analysis of the collected data. In this area it needs to be developed new innovative analysis methods that integrate several different types of data. The analysis work conducted today needs to be analyzed to identify missing functionality and tasks that can be automated. Analysis methods and guidelines also need to be developed and tested. This is a large research area that can be attacked from many different perspectives.

9 References

[1] NIKSUN, www.niksun.com

[2] SilentRunner, www.silentrunner.com

[3] Forensic mailing list at Securityfocus, forensic@securityfocus.com

[4] Focus-IDS mailing list at Securityfocus, focus-ids@securityfocus.com

[5] Casey, E. ed: *Handbook of Computer Crime Investigation*, Academic Press, 2001

[6] Shadow, www.nswc.navy.mil/ISSEC/CID/

[7] Guidance Software, www.encase.com

[8] The Coroners Toolkit, www.porcupine.org/forensics/tct.html

[9] Allen, Julia, et al.: *State of The Practice of Intrusion Detection Technologies*, (CMU/SEI-99/TR-028). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.

[10] *Information technology – Code of practice for information security management*, ISO 17799 part 1, 2000

[11] M. Bernaschi: *REMUS: A Security-Enhanced Operating System*, ACM Transactions on Information and System Security, Vol. 5, No. 1, February 2002, Pages 36-61

[12] Snare, http://www.intersectalliance.com/projects/Snare/Documentation/index.html

[13] Snort IDS, www.snort.org

[14] Network time protocol homepage, http://www.eecis.udel.edu/~mills/ntp.htm

[15] D. Brezinski and T. Killalea: *Guidelines for Evidence Collection and Archiving*, RFC3227, February 2002

[16] Paul E. Proctor: *The Practical Intrusion Detection Handbook*, Prentice Hall PTR, New Jersey, 2001

[17] John Tan: *Forensic Readiness*, http://www.atstake.com/research/reports/acrobat/atstake_forensic_readiness.pdf, July 2001

[18] IOCE: *Guidelines For Best Practice In The Forensic Examination Of Digital Technology*, http://www.ioce.org/2002/ioce_bp_exam_digit_tech.html, 2002

[19] E. Spafford, D. Zamboni: *Intrusion detection using autonomous agents*, Computer Networks 34, 2000, Pages 547-570

[20] C. King, E. Osmanoglu, C. Dalton, *Security Architecture Design, Deployment and Operations*, chapter 4, McGraw-Hill Osborne Media, 2001

[21] Psionic HostSentry, http://www.psionic.com/products/hostsentry.html

[22] Psionic ClearResponse, http://www.psionic.com/products/clearresponse.html

[23] Codefactory's IDXP-implementation, http://idxp.codefactory.se/

[24] Beep homepage, http://beepcore.org/beepcore/home.jsp

[25] B. Feinstein, G. Matthews, J. White: *The Intrusion Detection Exchange Protocol (IDXP)*, draft-ietf-idwg-beep-idxp-07, Internet draft, July 2002

[26] D. Curry and H. Debar: *Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition*, Internet draft, draft-ietf-idwg-idmef-xml-06, June 2002

[27] Homepage IDWG, http://www.ietf.org/html.charters/idwg-charter.html

[28] Marshall T. Rose, *BEEP The Definite Guide*, O'Re

[29] openSSL project, http://www.openssl.org/

[30] *G8 Recommendations on Transnational Crime*, Endorsed by G8 Ministries of Justice and Interior. Mont-Tremblant, May 2002. http://www.g8j-i.ca/english/doc1.html

[31] Council of Europe: *Convention on Cybercrime,* November 2001, http://www.conventions.coe.int/Treaty/EN/CadreListeTraites.htm

[32] H. Lipson: *Tracking and Tracing Cyber Attacks: Technical Challenges and Global Policy Issues* (CMU/SEI-2002-SR-009), CERT Coordination Center, November 2002

[33] Peter Sommer: *Intrusion Detection Systems as Evidence*, Recent Advances in Intrusion Detection – RAID 98, Louvain-la-Neuve, Belgium, September 1998

[34] J. Danielsson: *Project Description A system for collection and analysis of forensic evidence*, Application to NFR, April 2002

[35] IOCE: *G8 Proposed Principles For The procedures Relating To Digital Evidence*, http://www.ioce.org/G8_proposed_principles_for_forensic_evidence.html, 2002

[36] M. Ranum: *Some Tips on Network Forensics*, Computer Security Institute, 198 (September 1999): 1-8

[37] M. Bishop, C. Wee and J. Frank: *Goal-Oriented Auditing and Logging*, http://seclab.cs.ucdavis.edu/papers/tocs-96.pdf, 1996

[38] US Department of Defense: *DoD Trusted Computer System Evaluation Criteria*, (The Orange Book) DOD 5220-22-STD, 1985

[39] J. Kornblum: *Preservation of Fragile Digital Evidence by First Responders*, Air Force Office of Special Investigations, Digital Forensic Research Workshop, August 2002

[40] E. Lundin and E. Jonsson: *Privacy vs. Intrusion Detection Analysis*, RAID 1999, Indiana, USA, September 1999

[41] Computer Forensics Tool Testing (CFTT) Project Web Site, http://www.cftt.nist.gov/index.html

[42] P. Stephenson: *The Application of Intrusion Detection Systems in a Forensic Environment (Extended Abstract)*, Oxford Brooks University, School of Computing and Mathematical Sciences, Oxford, UK, RAID 2000, Toulouse, France, October 2000

[43] T. Perrine and A. Singer: *New Paradigms In Incident Management*, New Security Paradigms Workshop 2000, Cork, Ireland, September 2000

[44] P. Stephenson: *Intrusion Management: A Top Level Model for Securing Information Assets in an Enterprise Environment*, Proceedings of EICAR 2000, Brussels, Belgium, March 2000

[45] B. Fraser: *Site Security Handbook*, RFC 2196, September 1997

[46] V. Broucek and P. Turner: *Bridging the Divide: Rising the Awareness of Forensic Issues amongst System Administrators*, School of Information Systems, University of Tasmania, Australia, 2002

[47] Ming-Yuh Huang, Thomas M. Wicks, *A Large-scale Distributed Intrusion Detection Framework Based on Attack Strategy Analysis*, RAID 98, Louvain-la-Neuve, Belgium, September 1998

[48] D. Andersson, M. Fong, A. Valdes: *Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis*, http://www.sdl.sri.com/users/valdes/ndss01sp1.pdf

[49] J. Yuill, S. Wu, F. Gong and M. Huang: *Intrusion Detection for an On-Going Attack*, RAID 1999, Indiana, USA, September 1999
[50] Common Vulnerabilities and Exposures, http://cve.mitre.org/
[51] Bugtraq vulnerability database, http://online.securityfocus.com/bid/bugtraqid/

Appendix A User documentation

A.1 Installation

To install the system you should first obtain the source. Put the source in a directory. The source code has the following subdirectories:

Directory	Description
\	Root-directory of the distribution. Contains general library code, and documentation.
\adapters	Source-code for the adapters.
\agent	Source-code for the agent.
\coordinator	Source-code for the coordinator.
\log_server	Source-code for the log server.
\messages	Contains the message formats.
\test	Test-programs for the various “libraries”.

Run the command ‘make’ in the root-directory of the distribution. If everything went ok, the following binaries should be present:
agent\agent
coordinator\coordinator
log_server\log_server
adapters\auditd.so
adapters\pipe_adapter.so
adapters\run_adapter.so
adapters\snort_adapter.so

A.2 Usage

This system is actually not ready to be used yet, but the adventurous could try the following:

The system contains three main modules:

The agent is the module that controls the logging locally. The agent loads adapters according to which targets that one wants to supervise. The configuration of the agent and the adapters is given in agent.conf.

The log server receives all the log messages from the agents, and logs them. At the moment the log server just writes everything to a text file. The configuration of the log server is given in log_server.conf.

The coordinator is responsible for processing alerts, and sending collection orders to the various agents. At the moment it’s only possible to use one agent per coordinator. The configuration of the coordinator is given in coordinator.conf.

The configuration files have to be in the same directory as corresponding binary.

A.3 Format of the configuration files

The general format of all configuration files:

Reserved tokens:

: Marks the rest of the line as a comment.
$: Marks the beginning of an inserted variable. The variable must be in parentheses. Ex: $(MY_VAR). Only reserved if it is followed by a '('.
(: Used for variables.
) : Matches an '('.
{ : Marks the beginning of configuration statements that go over several lines, like for example the configuration of adapters.
} : Matches an '{'. Not allowed to be on the same line as an unescaped '{'

The following syntax-rules apply to the configuration files:

Two lines is considered as one line, if the first one ends with a '\'.

At the moment it's not allowed to have configuration statement blocks where both the opening bracket '{' and the closing bracket '}' are on the same line.

All non-empty, non-comment lines have to start with a keyword. The keyword marks the beginning of a configuration statement, and tells how the line (or block for statements that spans more than one line) should be parsed. Some keywords are general, while others depend on what kind of configuration file that is being parsed.

General keywords:

var : Defines a variable.
include: Includes another configuration file with the same format.
set : Marks an definition of a program variable.

A.4 Implemented adapters

At the moment four adapters are implemented: auditd, snort-adapter, pipe-adapter and run-adapter.

Auditd is an adapter that connects to SNARE's system call logging kernel module.

Snort-adapter uses the Unix-socket output plug-in in Snort. This plug-in writes snort-alerts in raw format (as a struct) to /dev/snort socket that the Snort-adapter creates, and reads from.

Pipe-adapter reads data from a pipe, at the moment hard coded to TEST_PIPE located in the same directory as the agent binary. The data is sent to the log server as strings. Pipe-adapter can be used in conjunction to all logging systems that lets you pipe the logs. An example on this is the apache web server and syslog.

Run-adapter can run Unix-commands. At the moment the run-adapter is hard coded to run the who command.

A.5 How to write your own adapter

It's pretty simple to make your own adapter. You need to implement the API defined in adapter.h:

```
typedef struct _adapter
{
  char id;
  int (*start)();
  int (*stop)();
  int (*reconfigure)();
  /* int (*shutdown)(); */
} adapter;

typedef int (*t_send_msg) (message* msg);

adapter* init(char* config, t_send_msg send_msg);
```

It means that you've got to implement an init function of the above format that returns an adapter struct. This function will be called from the agent. Further the function pointer start must point to the function that starts the adapter, stop must point to the function that stops the adapter, and reconfigure must point to the function that handles messages from the collection coordinator, for example about refined logging. Shutdown function is not yet implemented, but is meant to be a function the collection agent calls before it's shut down, to let its adapters to clean up. If some of these functions aren't defined, the corresponding function pointers should point to NULL.

The init-function is called from the agent-program. The parameters given are a text-string that contains the configuration of the adapter, and a pointer to a function. The function is of the defined function type t_send_msg, and is defined in the agent. The adapter calls this function when it wants to send some messages back to the agent.

The message structure is defined in message.h. To put it simple the adapter should build an alert message if it wants to reach the collection coordinator, or it should build a log message if it wants to send the message to the log server. The adapter then puts the message in a struct, or a string, depending of what format it chooses to use, and give the format and type of message in the header. Be aware that the structure that the pointer is referenced to is freed by the message handler in the agent, and that the pointer is used there.

For examples on simple adapters, take a look at the source code of the snort-adapter, or pipe-adapter.