Reliable and Tamper Resistant Centralized Logging in a High Availability System

- An Investigation on Ericsson SGSN-MME

Master of Science Thesis

ELIN KÄLLQVIST
JANNY QUACH LAM

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden, August 2012

Reliable and Tamper Resistant Centralized Logging in a High Availability System
An Investigation on Ericsson SGSN-MME

ELIN KÄLLQVIST
JANNY QUACH LAM

Examiner: Magnus Almgren

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden, August 2012

Abstract

Data logging is a critical activity and the foundation for several information security related activities, including intrusion detection, forensics and event reconstruction. Most often, computer systems where logs are collected consist of a network of connected machines. In such a network, it is often desirable to have an overview of how events are related between machines. Thus, having a central log server to which other machines can send log data for processing, correlation and analysis is important.

However, for the log data to be useful it must be trustworthy. To provide such trust, two important properties must be fulfilled, namely reliable and secure transmission, and tamper resistance.

The purpose of this project was to investigate how reliable and tamper resistant centralized logging can be acheived in Ericsson's SGSN-MME product. SGSN-MME is a high availability system where two components are redundant. To provide a trustworthy centralized logging in such an environment, failure of redundant components must also be taken into consideration.

To investigate whether and to what extent it is possible to achieve reliable transmission, two syslog applications, rsyslog and syslog-ng, were evaluated. To support the evaluation, a XEN-based lab system representing the SGSN-MME was designed. The different parts of SGSN-MME were simulated by virtual machines. Component redundancy was achieved by using open source software similar to what is used in the real SGSN-MME. To benchmark the two applications, seven test cases representing different failure scenarios were defined and executed.

The test results show that it is possible to achieve zero loss transmission for most of the test cases for both rsyslog and syslog-ng. To achieve zero message loss it is necessary to use both an application based transport protocol and a reliable disk buffer.

Tamper protection requirements have been determined by means of threat modelling. It is assumed that several users on SGSN-MME have root access. Therefore, it is concluded that the most promising solution is to make use of a cryptographic hardware device for encryption key storage.

Keywords: data log, syslog, rsyslog, syslog-ng, centralized logging, high availability, reliable delivery, reliable transmission, secure storage, data integrity

Contents

Abbreviations

DA	Disk-Assisted memory
FSB	File Server Board
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
NCB	Node Controller Board
ANCB	Active NCB
NFS	Network File System
PE	Premium Edition
RELP	Reliable Event Logging Protocol
RFC	Request for Comment
RLTP	Reliable Log Transfer Protocol
SGSN-MME	Server GPRS Support Node and Mobile Management Entity
SIEM	Security and Information Event Management
TCP	Transmission Control Protocol
TPM	Trusted Platform Module
UDP	User Datagram Protocol
VIP	Virtual IP-address

1 Introduction

Computer data logging is the process of recording information about events that occur in a computer system. The output of logging is a set of log entries where each one includes the characteristics of an event together with an associated time value. Logging information can among others be used for traceback, event reconstruction and discovery of digital evidence [1].

Since the log data is used for such purpose is it crucial that the information is trustworthy and authentic, in turn requiring computer systems with good protection against intrusions. The system needs to be protected against digital tampering because there are several reasons why someone would like to manipulate the log entries. For example, a misbehaving user may want to delete the traces of his actions or an intruder may want to remove his traces.

A technique that significantly increases the protection against tampering is to store the log entries on a remote log server. This forces an intruder to compromise two systems instead of one. The log system can also implement more security mechanism than a production system. A remote log server is also often desirable in distributed systems, where log entries from several machines can be collected in one common log file. Other programs and users then only need access to a single machine for all information [2].

However, forwarding log entries over the network to a central log server raises new complex problems. The trustworthiness must still be ensured and it is crucial that one can ensure that the transmission over the network is secure and reliable, as no messages should be lost. A high availability system with redundant log servers complicates remote logging even further. In such a system, no messages should be lost even though the primary server may fail and the secondary have to take over.

One such high availability system is the Ericsson SGSN-MME product which is one of the key components in a telecommunication system. A centralized logging infrastructure where log entries from all blades in the SGSN-MME are collected in a common location would simplify the log management of the product greatly.

1.1 Purpose

The purpose of this project is to investigate how centralized logging can be achieved in the Ericsson SGSN-MME product without message loss during transmission and how tampering of the stored log messages can be prevented.

1.2 Scope

The project will focus on performing tests of two syslog applications, namely *syslog-ng* and *rsyslog*, regarding reliable delivery in a virtual lab network representing SGSN-MME. A survey of different solutions for secure file storage will also be performed but these will not be tested in the virtual lab system.

1.3 Report Outline

Chapter 2 presents related works in the following areas; data logging, reliable delivery of data and integrity of stored data.

Chapter 3 covers the technical background. An overview of Ericsson Packet Core Network is presented and is followed by a more detailed description of SGSN-MME, a component of the network. A computer data logging protocol called syslog is introduced and some cryptographic techniques are explained.

Chapter 4 describes the workflow of finding a solution for secure centralized logging in SGSN-MME. Identification of requirements is done followed by a description of the threat modeling used to find the threats in the system. The test cases needed for evaluation are described.

Chapter 5 describes the design and implementation of the virtual lab system regarding both the software that is used and the flow of log transmission to achieve centralized logging.

Chapter 6 describes in more detail how the tests were performed in the lab system.

Chapter 7 presents and discusses the test results of the transmission phase achieved by performing test cases in the lab system. A short survey of secure file storage is also presented with a discussion how to prevent tampering after transmission.

Chapter 8 presents potential future work.

Chapter 9 states conclusions based on studies and test results.

2 Related Work

This chapter presents interesting works related to this project. The areas discussed are data logging, reliable data transmission and tamper resistance of stored data.

2.1 Data Logging

Taxonomies are a good starting point in order to learn more about logging mechanisms. There exist several taxonomies which focus on logging and among these are the work by Schroeder [3], Albari [4], and Larson et al. [5]. These focus on several dimensions for classification, including mechanism configuration, sensor types, collected information and system architecture.

The *National Institute of Standards and Technology* (NIST) provides technical recommendations for the US economy and public welfare. They develop management standards and guidelines for cost effective security of sensitive information in federal computer systems. In *Guide to Compuer Security Log Management* [6] NIST states recommendations that would provide a more efficient and effective log management for the federal departments and organizations.

The report [6] states the importance and usefulness of log information as well as the challenges in log generation, storage and protection. The key solution for a better log handling includes establishment of polices and procedures, prioritization of log management, creation and maintenance of a secure logging infrastructure and also the importance of adequate support for all staff with log management responsibilities. The report gives clear and useful information about both technical details of implementation as well as which planning and preparatory actions an organization should perform to achieve effective and secure logging.

Saxena et al. desribes the importance of protecting the logs for computer forensics and the difficulties of log management [7]. In a system with multiple machines it is important that the machines have the same time and that the same event on different machines produces the same content for the logs. The volume of logs can be large and can be reduced by filtering unnecessary logs. Administrators often consider log analysis as a low-priority task and it is often done after a problem has been identified when log analysis can be used for preventing the problem.

Hall discusses the importance of a secure logging server to prevent attackers from hiding their tracks [8]. The paper presents basic security concepts to create a secure Linux logging system by using a remote log server. For example a password should be nearly impossible to guess and the log server should have as few software installed as possible. Hall states that many software have unknown security flaws and if these are found the security of the log server could be compromised.

In 2009, Accorsi [9] made an extensive survey of secure logging protocols. Even though this was only three years ago many of those protocols have not been further developed and were not seen as alternatives in our implementation. However the author had some very good discussions regarding requirements of logging protocols. The large difference between the survey and our investigation is that we work with a specific system and therefore can set more specific requirements. Accorsi's requirements were strict but rather general so that they could work in all systems. Accorsi state that no protocol he surveyed at that time fulfills all the necessary conditions to be used as admissible evidence.

The SGSN-MME node and several other network components in Ericsson's packet core network already use the syslog facility [10]. Among the two most well known syslog applications are *rsyslog* [11] and *syslog-ng* [12]. These are actively developed and maintained, both also offer features for reliable centralized logging and tamper protection of data.

An alternative for syslog is the ongoing Fedora project called *journald* [13, 14]. Journald produces log messages in binary which is more difficult to tamper than log messages in clear text as in the traditional syslog. It can not replace syslog for all Unix or Unix-like systems since it requires the system to run the startup daemon systemd. No Linux distribution for embedded systems support systemd at this time and therefore it can not be used in SGSN-MME. Systemd knows all processes in the system by using cgroups to track service processes and therefore journald provides process authentication of log messages since no process can pretend to be another process. Process authentication is not in focus in this project.

2.2 Reliable Transmission

Currently in SGSN-MME, data is collected on each board and stored in its own area on the FSB. A more suitable solution would be to have a centralized logging mechanism which uses only one log file. Several vendors provide centralized logging features as part of their SIEM (Security and Information Event Management) solutions [15, 16, 17]. However, a new infrastructure is not feasible to introduce to the SGSN-MME. Therefore, these are seen as references rather than real alternatives.

Tsunoda et al. state that the *Transmission Control Protocol* (TCP) can not guarantee reliable delivery of syslog messages [18]. TCP can only guarantee no packet loss as long as the network connection is kept. This is an important piece of information to our work. Since we require no message loss even during a failover, when the network connection is lost, another transport protocol must be considered.

2.3 Tamper Resistance of Data

In [19] a set of mechanisms for protecting logs using Unix file permissions is proposed. Since several users have root access to the SGSN-MME and not all these can be fully trustworthy the SGSN-MME system can be seen as an untrusted system which makes secure storage more difficult. In [20] a set of techniques for ensuring tamper resistance of log files is discussed, among these, hash chaining [21] and write-once techniques [22].

Furthermore Kelsey och Schneier [23] have presented a scheme for secure log storage on an untrusted machine. The scheme guarantees confidentiality and detectability of modified logs before the time when the machine was compromised. However some interaction with a trusted machine is needed and is therefore not an option in the current SGSN-MME.

Moving cryptographic computing such as hashing from software to hardware provides more protection against software attacks. One such mechanism is the *Trusted Platform Module* (TPM), a microcontroller chip, defined in ISO/IEC 11889 [24] which can be used to generate cryptographic keys, storage of keys, generate random numbers and perform cryptographic computing [25].

Accorsi [26] presents an architecture and the main components of a digital black box called *BBox* which provides authentic archiving of log entries for remote logging in distributed systems. BBox is based upon public key cryptography and a Trusted Platform Module. There existed no public implementation of BBox at the time when this report was written.

3 Technical Background

This chapter starts by presenting an overview of the Ericsson Packet Core Network. It continues with a more detailed description of SGSN-MME which is the network component where centralized logging is desired. Next is a description of the syslog protocol. The chapter ends with an introduction to some general cryptographic terms and methods.

3.1 Ericsson Packet Core Network Overview

Telecommunication is a complex area, including several networks, standards and protocols for phone calls, SMS, MMS, data traffic etc. This section will give the reader a brief overview of the communication parts that are relevant for this specific project. Focus will be on the Ericsson Packet Core Network which provides IP communication between a user equipment, for example a phone or computer, and an external network, for example Internet. Figure 1 shows the packet core network together with the surrounding radio networks and external networks. The radio networks provide wireless data communication to and from the user equipment. Ericsson Packet Core Network can be connected to the following radio networks standards; GSM (2G), WCDMA (3G) and LTE (4G). External networks can for example be the Internet, Local Area Network or operator's service network.

Ericsson Packet Core Network consists of two parts; the General Packet Radio Service (GPRS) network and the Evolved Packet Core (EPC) network. The GPRS network provides packet data services for GSM and WCDMA systems. The EPC provides services for LTE networks.

The Packet Core network consists of several nodes which together provide functions in the network. These nodes together with some of their connections are shown in Figure 1.

Serving GPRS Support Node (SGSN) is the key control node in GPRS and handles among others the establishment of connections between the user equipment and the external network. Mobile Management Entity (MME) is the key control node in the EPC network and handles the main functions there. Gateway GPRS Support Node (GGSN) is a gateway between the GPRS network and an external network. The Serving Gateway (SGW) routes and forwards the user packet data between the user equipment and the PGW. The Packet data network Gateway (PGW) is a gateway between the user equipment and the external network in the EPC network.

There are no strict borders for the nodes, such as wheter a specific node belongs to the GPRS or the EPC network when implemented in the customer's network. For example, the SGSN and MME nodes are implemented in the Ericsson product SGSN-MME which can be used with either SGSN functionality or MME functionality, or both [27].

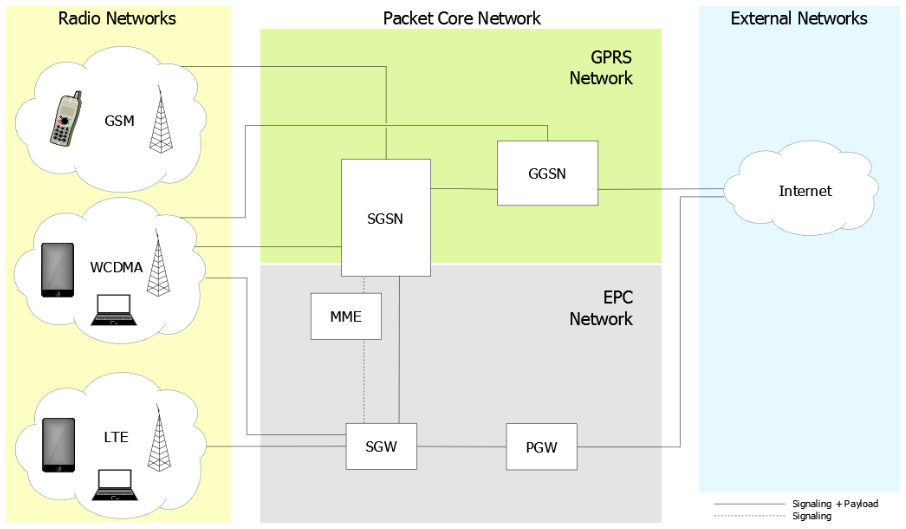


Figure 1: Ericsson Packet Core Network with surrounding networks [27].

The connections between the nodes are made clearer by the following example. Suppose a user wants to visit a website on her smartphone. First the phone signals the nearest Radio Base Station which is responsible for the transmission and reception of information over the radio network. The information is forwarded to SGSN. Among other functions this node ensures the proper authentication of the phone by checking a database holding subscription information. After a successful authentication SGSN forwards the IP packets to Internet through the GGSN.

3.2 SGSN-MME Overview

SGSN-MME provides IP packet services between the radio networks and external networks. One of the main services is IP-routing which is the determination of the best path to send packets through the network. Another service is mobility management, which keeps track of the possible changing location of the user equipment. SGSN-MME handles authentication of the user equipment and ensures that it is allowed to use the network.

An SGSN-MME node consists of a cabinet housing internal cabling, power distribution units, fan units and several circuit boards. Most boards have Intel processors running Linux operating systems. For communication purpose the boards are interconnected through a backplane and together they provide all features of the SGSN-MME. Two of the most important circuit boards are the NCB and FSB.

The *Node Controller Board* (NCB) is a circuit board that provides several central functions in the node.

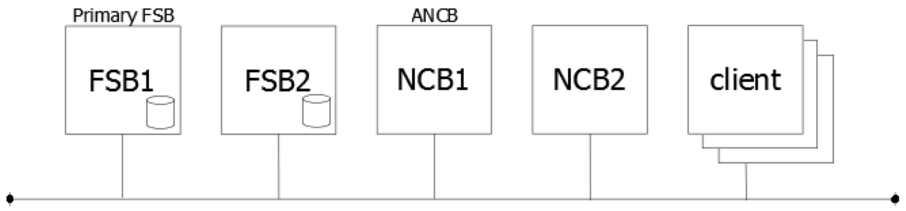


Figure 2: A simplified view of SGSN-MME.

The *File Server Board* (FSB) is the only board within the system that contains a physical hard disk and provides disk storage for all boards. Other boards can store data to their virtual local disk, which in reality is a part of the FSB hard disk [27].

The number of boards in SGSN-MME depends on the configuration of the node. Apart from the NCB and FSB, the name and functions of other boards are not important for this project and from now on they will be referred tp as *clients*.

SGSN-MME is designed as a *high availability* system [27]. Availability refers to the ability of users to access a system. If users cannot access resources of the computer system then it is said to be unavailable. If the SGSN-MME system becomes unavailable the whole network will stop working. High availability is a system design that ensures a certain and very high level of operational performance will be met during a predetermined period of time [28].

There are different ways to achieve high availability [28]. One method is to use redundant hardware, which is the solution used for the SGSN-MME. Both main components, the FSB and the NCB, are duplicated, so there is one active and one passive FSB, and one active and one passive NCB. The active board is the one in operation and the passive board is mirroring the active one's disk at all times. In this report the two NCBs are referred to as NCB1 and NCB2 where NCB1 is the *active NCB* (ANCB) when it is up and running. Similarly, the two FSB's are referred to as FSB1 and FSB2. The active FSB is referred to as the primary FSB, the passive as secondary FSB. Figure 2 shows the roles before an active board stops working. FSB1 has the primary role and NCB1 has the role of ANCB.

When an active board fails, a *failover* happens. All shared services migrate to the passive board that takes over the active role until the other board is functioning properly again. A *switchover*[1] is essentially the same operation as failover, but intentionally done by for example an administrator. The failover is automatic and not a planned operation. The difference is that a switchover allows the system to perform an orderly shutdown and therefore provides some time for completion of tasks. In a failover the system stops immediately as a

[1]At Ericsson the term switchover is used equivalently to failover.

system crash or a power supply stop. A *failback*[2] is the process of restoring the state so that the first board takes over the services and become the active one when it is up and running again [29]. Failover and failback are illustrated in Figure 3.

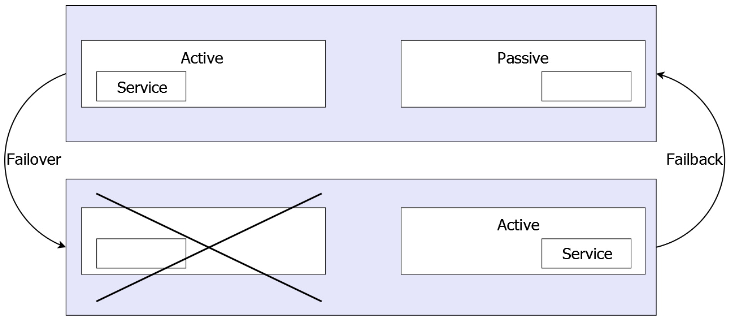


Figure 3: States before and after failover and failback [29].

3.2.1 Current Implementation of Log Handling

In the current SGSN-MME all boards store their logs on their local disk, which in reality is a segment of the FSB disk. This leads to several log files, more exactly, one for every board. This is inconvenient for the operators, because in case of a failure, they need to read several files to find the information of interest. The logs are processed by a default syslog application, syslog-ng. Syslog will be described in section 3.3 and syslog-ng will be described in section 4.2.

3.3 Syslog

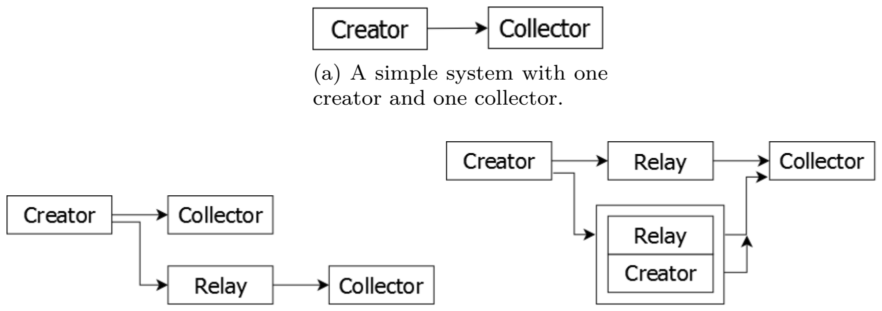
Syslog was developed in 1980s for the Sendmail-project [30]. With time many systems and devices started to use syslog to log system data, but no reference for the syslog protocol existed until the Internet Engineering Task Force (IETF) Request for Comment (RFC) 3164 was written in 2001 [31]. IETF RFC 5424 [10] was later written, describing the syslog protocol more specifically. Syslog is characterized by its simplicity and that is probably also the main reason why it is widely used. The syslog messages are written in clear text and are sent with User Datagram Protocol (UDP).

Syslog is a system that can generate, store and process meaningful log messages. It is both a communication protocol as well as a set of programs and libraries. The syslog applications, also called syslog daemons and often written *syslogd*, can vary greatly in terms of performance, security etc., but the protocol should be the same across all applications.

[2]SGSN-MME does not initiate failback. The active board stays active even if the other board comes up.

A syslog application can take one of three different roles. A *creator* generates the syslog content that will be contained in a message and the *collector* gathers syslog content for later analysis. A *relay* forwards messages it receives from a creator or a relay and sending them to a collector or another relay.

Three examples of syslog systems are illustrated in Figure 4. A simple system can consist of a creator sending its messages to a collector. A creator or a relay can also be configured to send the same message to multiple relays and collectors. A system can be configured to function as creator, relay and collector at the same time. For example relays can forward all or some of the received messages and also send internally generated messages.



(a) A simple system with one creator and one collector.

(b) A system with a relay.

(c) A relay can also be a creator.

Figure 4: Examples of syslog systems [10].

A syslog message has generally three parts; PRI, header and message. *PRI* is an eight bit number that represents both the *facility* and the *severity* of the message. The three least significant bits represent the severity value which indicates the importance of the message. Messages with severity value zero are most important. The other five bits represent the facility of the message, which indicates the application or operating system component that generated the message. Table 1 shows the meaning of the different values in severity and facility. The *header* part contains a *timestamp*, which is the date and time the message was created, and *hostname or IP address* of the device. The *message* part has two fields called *tag* field and *content* field. The tag field indicates the name of the program or process that generated the message and the content contains the details of the message.

All of the fields mentioned above are commonly found in a syslog message, but not always present depending on implementation and configuration. Fields can for example be used for filtering and processing. Messages from a certain process or from a specific host can be particularly interesting and desired to be stored at some other destination than the others. These messages can later be used for troubleshooting or analyzed by an administrator to discover intruders in the system.

#	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediatly
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

#	Facility
0	Kernel messages
1	User-level messages
2	Mail system
3	System daemons
4	Security/authorization messages
5	Messages generated internally by syslogd
6	Line printer subsystem
7	Network news subsystem
8	UUCP subsystem
9	Clock daemon
10	Security/authorization messages
11	FTP daemon
12	NTP subsystem
13	Log audit
14	Log alert
15	Clock daemon
16-23	Local use 0 - 7

Table 1: Severity and facility values with their numerical codes in syslog [10].

3.4 Cryptographic Terms

This section gives some basic knowledge of the cryptographic terms and functions that are important for the understanding of this report. It gives a brief introduction to public key cryptography, hash functions and digital signatures.

Modern cryptographic techniques can be divided into two parts, symmetric key encryption where both encryption and decryption are made with identical keys and asymmetric cryptography which requires a pair of keys, one for encryption and one for decryption. Asymmetric key cryptography is also known as public key cryptography since one of the keys is public. Public key cryptography methods can be used both for confidentiality and authentication.

When *public key* cryptography is used for confidentiality purpose, the public key is used for encryption and the private key for decryption. For example suppose Alice wants to send a secret message to Bob. Then Alice encrypts the message with Bob's public key and sends the message to Bob. Then Bob decrypts the message with his private key. Since Bob is the only one knowing the private key, he is the only one that can decrypt the message.

Sometimes the most important property is not to keep the message secret but to ensure that it has not been altered. This can be ensured by public key cryptography and hash functions. A cryptographic *hash function* is a one way function which takes input data with arbitrary length and produces a fixed-length output, a hash value. To ensure that a message is the same as the original, one first computes a hash value of the original message and later computes the hash value of the message one wants to ensure is identical. If the two hash values are equal, then the messages are identical. A cryptographic hash function has two main properties. It should be computationally infeasible to regenerate the message for a given hash value, and it should be computationally infeasible to find two different messages which produce the same hash value.

Public key cryptography and hash functions can be combined to form *digital signatures*. These can be used to ensure that the message has not been altered and that the sender is the real one. A sender creates a hash value of the message and encrypts it with his private key. He sends both the message and the encrypted hash value to the receiver. The receiver decrypts the hash value with the sender's public key. He computes a new hash for the message and compares it to the received hash. If they are identical he knows that no one has altered the message and that the receiver is the one he claims to be. He also knows that the only one that could have created the signature is the owner of the private key, thus achieving non-repudiation [32].

4 Methodology

This chapter describes the project workflow used for investigating how centralized logging can be achieved. The project started by getting familiar with SGSN-MME and relevant technologies, and thereafter an identification of requirements for secure centralized logging in SGSN-MME was done. Centralized logging can be divided into two phases; transmission and storage. A research of existing syslog applications was done and due to the time limit and the set requirements, two syslog applications were chosen to be tested. A virtual system representing SGSN-MME was developed to evaluate solutions for the transmission phase. To benchmark different solutions in the transmission phase, a set of test cases was defined and used in the lab system. The storage phase is harder to test, therefore a process called threat modeling has been used to identify the threats in the system for the stored log messages. Thereafter, different methods have been discussed.

4.1 Identification of Requirements

The purpose of this project is to investigate how secure centralized logging can be achieved. Centralized logging is reached by letting a log collector receive log entries from creators or relays in a network and append all entries to one single log file. This project focuses on creating a log file storing log entries defined as security events.

Security events are syslog messages with facility value auth or authpriv and with a severity value higher than debug.

Log messages created by the authorization system can for example be someone trying to logon a machine, whether it is a success or a failure. The numerical codes of the facility values are 4 and 10 in Table 1.

Centralized logging can be separated into two phases; *Transmission phase*, when log messages are sent over the network from creator to collector. *Storage phase*, log collector storing the received log messages. The following sections will describe the requirements of the SGSN-MME regarding log handling in both phases.

4.1.1 Transmission Phase

There are several aspects to take into consideration regarding the security requirements during the transmission phase. *Creator authentication* is the assurance that the logs are sent from an authorized creator. *Process authentication* is the assurance that the logs are sent from an authorized process. *Message confidentiality* ensures that the transmitted data are not available to unauthorized users. *Message integrity* is the assurance that the transmitted data can not be modified. *Message uniqueness* is the assurance that log messages are logged just once. *Reliable delivery* ensures that any data sent must reach the collector [9].

All of these requirements are relevant and should be considered when implementing centralized logging in SGSN-MME. Due to time constraints, this project has restricted the requirements to mainly focus on reliable delivery. In this project we use the following definition.

Reliable delivery of log messages means that any log message created must reach the collector and be stored in a stable memory.

4.1.2 Storage Phase

During the storage phase one can categorize the important security aspects as follows. *Entry integrity* ensures that log entries can only be altered in an authorized manner. *Entry confidentiality* ensures that log entries are encrypted so only authorized users can read them [9].

In this project, the priority after reliable delivery is integrity in the storage phase. In SGSN-MME no one should be authorized to modify the existing log entries.

Log integrity is achieved if it is impossible to modify or delete log entries. The log daemon process is the only one that should be authorized to add entries to the end of the file.

This is an ideal requirement and if it can not be fulfilled one should at least be able to discover that someone has made changes to the log file.

4.2 Choice of Logging Application

This section discusses the choice of logging applications that have been investigated in this project. Focus has been on applications similar to the logging system in the SGSN-MME which can easily be adopted with minimal changes.

Since the most boards in SGSN-MME run Linux, all logging applications that do not run on this platform can be ruled out. The syslog protocol described in section 3.3 is standard in many Unix and Unix-like systems. However, it was not developed to provide secure logging as it transmits and stores messages in clear text and uses UDP for delivery. This has led to a number of extensions of the traditional syslog system where most extensions have focused on the transport of data. Most extensions support the use of TCP instead of UDP.

By using TCP instead of UDP as the underlying transport protocol for syslog, the message loss can be decreased. However messages can still be lost due to several reasons as mentioned in section 2.2. One is that the reliability is only assured as long as the network connection is established. Another reason for package loss can be that TCP only guarantees that the destination host receives the packets, not that they are received by the destination application. If a log message is acknowledged but the machine crashes before the log application has received it, the message will be lost. To accomplish a fully reliable transmission where no messages are lost a retransmission mechanism on the application layer is needed [18].

The developers of the syslog applications *rsyslog* and *syslog-ng* have realized this problem and offer a special protocol for transport of log messages that provides an acknowledgement at the application level. In rsyslog it is called *Reliable Event Logging Protocol* (RELP) and in syslog-ng *Reliable Log Transfer Protocol* (RLTP). Another feature the applications have is buffering of log messages if the log server is unreachable which is very important if reliable delivery is desired [11, 12].

Both rsyslog and syslog-ng provide special protected storage of log entries, which is interesting for the integrity part of this project. The storage technology is based on hash chaining [11, 33]. This method prevents injection, deletion, or modification of the stored data. When adding new data to the log file, the data is hashed together with the hash value of the previous added data. So the chain consists of several hash values where each is dependent on the previous one. If an intruder changes data in the middle of the log file and recomputes the hash for it, this can be discovered since the next hash value will then be wrong. To be able to verify the integrity of the logs the last hash needs to be saved to an external location for later validation of the hash chain.

Rsyslog offers this protected store implementation in the independent package *LogTools*. LogTools uses the file format *LogStore* to store log messages, and consists of two tools; LogReader and LogWriter. LogWriter is used to write log messages to a LogStore. The tool processes text files which need not necessarily be log messages. LogReader is the inverse tool to LogWriter and is the utility that reads messages from a LogStore and presents them in a more readable form. LogTools is an active project and has the intention to enhance secure storage with digital signatures to protect the hash chain [11].

Syslog-ng Premium Edition (PE) provides protected storage in so called *logstore* files. New log entries are collected to a chunk and each chunk is hashed together with the previous hash value. The application can encrypt the chunk using different algorithms as well as compressing and timestamping them. The application provides a tool called *lgstool* for reading the encrypted messages [33].

4.3 Virtual Lab System Design

To accomplish an evaluation of possible solutions, a virtual lab system representing a simplified SGSN-MME system is developed. To be able to test all cases described in section 4.4 a minimum lab system requires two redundant NCB modules, two redundant FSB modules and one client. The virtual lab system is used to configure different logging applications and to test if they can achieve zero message loss during the tests. The design of the lab system is further described in section 5.1.

4.4 Test Case Design

Test case design is the procedure of writing the tests that are intended to be used to evaluate possible solutions. Table 2 shows test cases developed for the SGSN-MME system. The goal for all tests is that the reliable delivery property should hold. For success, all tests should be performed with zero message loss.

The potential problem with test number 1, 2 and 3 is that logs can be lost during the time when no board has the role of ANCB. Similarly, logs may be lost in test 4, 5 and 6 during the time when no board is the primary FSB. In test 7, log messages can be lost if a log client crashes before all generated messages are sent. These messages should be sent when the log client is up and running again. Test nr 8 and 9 are combinations of the previous ones.

#	Test	System state before test
1	NCB switchover	All boards running
2	NCB failover	All boards running
3	NCB failback	NCB1 stopped, other running
4	FSB switchover	All boards running
5	FSB failover	All boards running
6	FSB failback	FSB1 stopped, other running
7	Client crash	All boards running
8	NCB failover & FSB failover	All boards running
9	FSB failover & NCB failover	All boards running

Table 2: Test cases for the transmission phase in SGSN-MME.

4.5 Threat Modeling for Log Storage

Threat modeling is a process used to identify the existing threats provided a specific environment, assets and actors. The assets in a system are the devices, software, data and so on for which the security requirements are set. The assets in SGSN-MME are the log files holding valuable information and the log daemons creating, forwarding and receiving the logged events, shown in Table 3a.

After the identification of the assets, the next step is to define and describe the environment and the entry points from surrounding systems. Figure 5 shows the system environment and the networks from where one can access the SGSN-MME which is holding the assets. Customer operators can access the NCB board in the node from the Operation and Maintenance (O&M) network. Also Ericsson employees can connect to the O&M network from the Ericsson network via a *Remote Service Gateway* (RSG). From RSG they can also connect to the Backup & Restore network from which they can gain access to the FSB board.

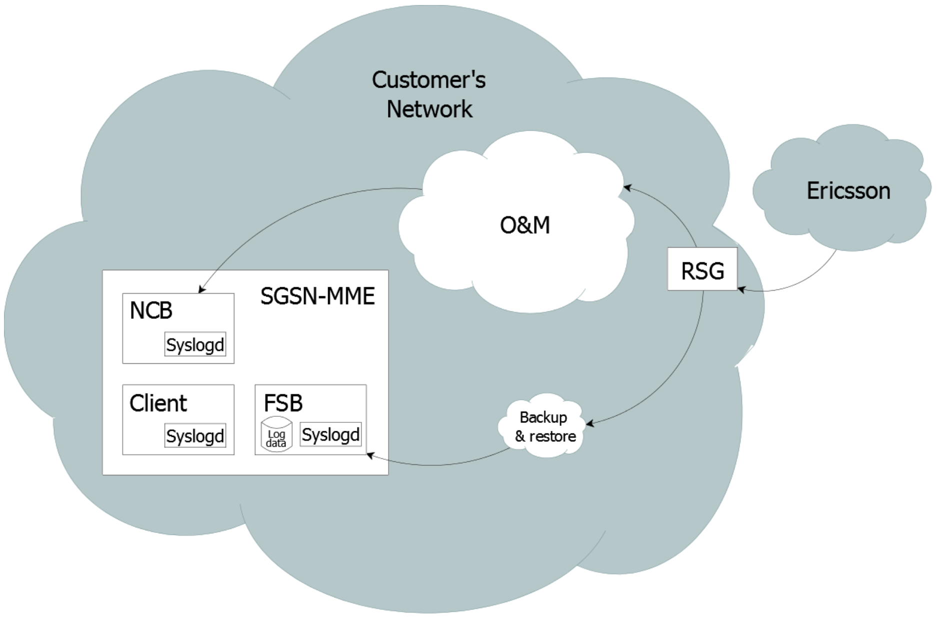


Figure 5: Entry points to the SGSN-MME holding the assets.

Step three in the threat modeling is to identify actors that may have interest in gaining access to, modifying or destroying the assets. Protection against external attackers should be taken care of by for example firewalls and are not considered in this project which will focus on insiders, that is authorized users of the system. Table 3b shows the identified actors in our system which are the ones that can access the assets and what kind of access they can obtain. All of these actors can gain both normal and root access to the system. The root user in a Unix or Unix-like system has access to all commands and files while a normal user has less privileges.

After the identifications of assets and actors, the identification of threats is done by defining possible actions the actors can do to the assets in the system. A list of threats found in the SGSN-MME system regarding the previous defined assets are listed in Table 3c. Only the four first threats in the table concerns the integrity requirement in the storage phase, the others are potential future work. As seen in Table 3c, a root user can perform all actions defined as threats for SGSN-MME and all of the actors defined in Table 3b can obtain root access.

This indicates that if not all users with root access are fully trusted then the system is not secure and the assets need to be protected from the root user as well as other users in the system.

The final step in a threat modeling is to identify suitable solutions to lower or eliminate the threats. This step is discussed in section 7.2.

Assets
Log file
Log daemon

(a) Identification of assets.

Actors	Access
Ericsson employees during support	normal, root
Customer operators	normal, root

(b) Identification of actors and their access rights.

Threat description	Assets	Access
Remove log file	Log file	root
Remove data from log file	Log file	root
Add data to log file	Log file	normal, root
Modify data in log file	Log file	root
Flood the log (DoS)	Log daemon	normal, root
Starve the daemon	Log daemon	normal, root
Kill the daemon process	Log daemon	root
Break network communication	Log daemon	root
Flood the NFS-service	Log daemon	normal, root
Fill up the disk	Log daemon	root

(c) Identifiction of threats.

Table 3: Threat modeling.

5 Implementation and Design

This chapter describes the lab system used as test environment and also how centralized logging can be achieved by two different syslog daemons; rsyslog and syslog-ng.

5.1 Lab System

A test environment simulating the real SGSN-MME is needed in order to evaluate different methods for secure logging. All boards in SGSN-MME are not relevant for this project and are therefore not included in the lab system. The simplified SGSN-MME consists of the important components FSB and NCB, and one other component called client which represents the other possible boards. An overview of the lab system can be seen in Figure 6.

The test system is developed in a Linux environment and all components are simulated by virtual machines. The *Xen hypervisor*[3] is used to create and manage the virtual machines. It is an open source program that supports multiple operating systems to execute concurrently on the same hardware [34]. The lowest guest *operating system* (OS) is in Xen terminology called Dom0 and is the only guest that has direct access to the hardware and therefore acts like a switch for the network allowing the other OS to communicate. Dom0 represents the backplane in the SGSN-MME. The other guests are unprivileged domains called DomUs which will represent the boards. Both the development environment Dom0 and all DomUs are running the Linux distribution Debian[4].

Since no boards but FSB have a physical hard disk, a distributed file system protocol called *Network File System* (NFS) is used to virtualize local hard disks for the other boards in SGSN-MME. NFS is a protocol that makes it possible for clients to access files on the server as they were located on their local disk [35, 36, 37]. In SGSN-MME, FSB is the NFS server and all other boards are NFS clients. Even though this is implemented in a similar way in the lab system, all virtual machines have a local disk to make it easier to discover the reason for possible message loss. The security events that are not stored in FSB are seen as lost, but if more or fewer messages are stored in the local disks of the other boards then it can give us a hint of why the messages disappeared. It is possible to configure the underlying transport layer for NFS to be UDP or TCP. UDP is used in the real SGSN-MME, but TCP has also been tested in the lab system.

As explained in section 3.2, the FSB and NCB boards in SGSN-MME are duplicated to achieve high availability. Therefore also the virtual machines representing these are duplicated. As seen in Figure 6, the lab system consists of five virtual machines namely FSB1, FSB2, NCB1, NCB2 and client. There are

[3]Xen hypervisor, `www.xen.org`
[4]Debian, `www.debian.org`

numerous online guides discussing how to set up a high available server consisting of two redundent machines [38, 39, 40]. The NFS server FSB was set up by using these guides. The redundancy features are accomplished by DRBD[5], Pacemaker[6] and Corosync[7]. These program packages are all open source and used because of their similarity to the software in the original SGSN-MME. The NCB will act as a log server in our system and was set up in similar manner to FSB, but since the boards do not have any disks in reality there is no need for DRBD.

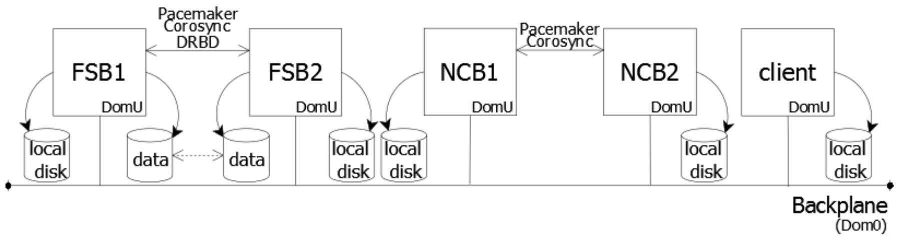


Figure 6: The lab system representing a simplified SGSN-MME.

The *Distributed Replicated Block Device* (DRBD) is responsible for the mirroring of the FSB disks. All data blocks that are written to the primary disk is also immediately written to the secondary disk. Synchronous mirroring is used which means that a write to disk is reported successful only when the write is completed in both the primary and secondary machine. This together with the feature that data only can be accessed from the primary node is necessary to achieve complete consistency between the disks [41].

Pacemaker is used to detect when the active machine fails and will initiate a failover to the passive machine. It is also responsible for starting all shared services on the new active machine. Pacemaker is combined with a cluster infrastructure called *Corosync*, which handles the messaging between the redundant machines. The messages are UDP packets sent out once every half a second from one machine to determine the status of the other machine. Pacemaker provides high level control of the cluster while Corosync provides low level inter-node communication and command framework [42].

One of the shared services managed by Pacemaker is the *virtual IP-address* (VIP). This service is used to make the redundancy of the FSB and NCB machines transparent to other machines in the system. A VIP is set up by telling Pacemaker the desired IP-address to be shared by the redundant machines. Pacemaker ensures that the IP-address belongs to the active machine, even in a failover. The two FSBs share one IP-address and the NCB1 and NCB2 also share one IP-address.

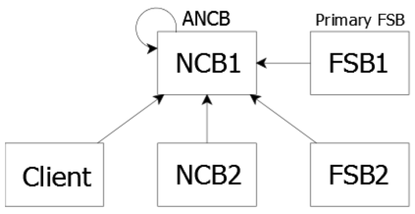
[5]DRBD, `www.drbd.org`
[6]Pacemaker with or without Corosync, `www.clusterlabs.org/wiki/Install`
[7]Corosync, `www.corosync.org`

Two other services handled by Pacemaker on FSB is the DRBD and NFS features. Pacemaker ensures that FSB2 starts all services when FSB1 goes down, and during a failback it stops the services on FSB2 and starts them on FSB1. Both NCB1 and NCB2 mount the FSB VIP and do not know that there are two machines.

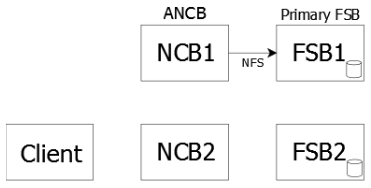
5.2 Log Transmission Design

One of the objectives in this project is to find a way to collect the security events of all boards in SGSN-MME and store them in a single file. In our test environment the local syslog daemon present on a client sorts all created syslog messages. All logs are stored locally and then security events are also forwarded to the log collector.

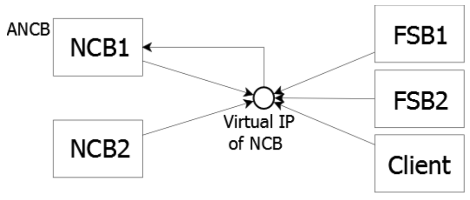
Figure 7 shows the desired flow of security events in SGSN-MME during normal operation. All boards in the system are creators and send their security events to the collector, also called log server. ANCB is the log server and works both as creator and collector in system. Illustrated in Figure 7a is the scheme of how the components in the system send their security events using the syslog protocol. Since only FSB have a real hard disk, ANCB will need to use NFS to store all log messages on FSB, illustrated in Figure 7b.



(a) All components are log clients using the syslog protocol to send their security events to ANCB. ANCB is both log client and log server.



(b) ANCB uses the NFS protocol to store the security events on FSB.



(c) Log clients do not know which of the NCB is active. All machines, even the ANCB, send their security events to the virtual IP of NCB.

Figure 7: The flow of the security events during normal operation.

Clients in the system do not see NCB1 and NCB2 but only NCB and are therefore configured to send their logs to the virtual IP-address of NCB. As Figure 7c shows, all clients including both NCBs are sending their security events to the NCB VIP which belongs to ANCB. Neither NCB nor other clients can see FSB1 and FSB2. FSB also has a VIP which belongs to the primary FSB.

In case of a failure of NCB1, NCB2 will take over the role as ANCB. The VIP is then owned by NCB2 and all logs are forwarded to NCB2, as shown in Figure 8. Figure 9 shows the scenario of an FSB failover. The FSB2 will become the primary FSB and therefore own the FSB VIP. Consequently, ANCB will send log messages to FSB2. The disks of FSB1 and FSB2 are mirrored at all times, so all security events will be stored in the same file.

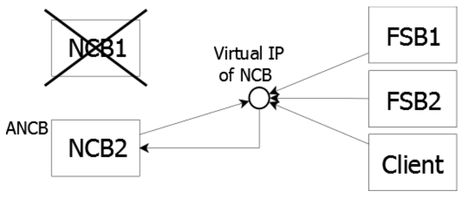


Figure 8: Log messages will be forwarded to NCB2 after an NCB failover.

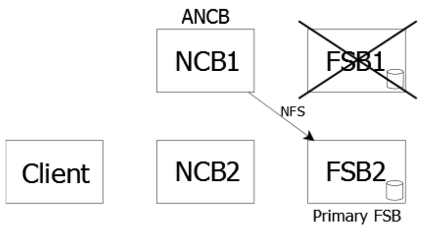


Figure 9: ANCB will store log messages to FSB2 after an FSB failover.

5.3 Implementation of Log Application

Rsyslog respectively syslog-ng are installed and configured in the lab system. Rsyslog is an open source implementation while syslog-ng has an open source edition and a premium edition (PE). Both applications are based on the basic syslog protocol, extended with features such as content-based filtering and option of using TCP for transport of logs. Rsyslog and syslog-ng also provide an extra transmission protocol to improve the known issues of TCP discussed in section 2.2. The protocols provide an acknowledgement at the application level and they attach a transaction number in the message and the server acknowledges all messages. If some messages are not acknowledged the client will resend them. The server may acknowledge the messages in any order. Both RELP and RLTP are built on top of TCP to benefit from the TCP timeouts features [43, 33]. RLTP is only available in syslog-ng PE.

5.3.1 Rsyslog

One rsyslog daemon version 6.3.10 (beta) is running on each machine in the lab system. This version supports *ruleset* for RELP. A ruleset is a set of rules where each rule consists of a filter and performs one or more actions when the filter evaluates to true. It is possible to have multiple rulesets to do different actions. The below example shows a simple ruleset named `remoteEvent` with only one rule that filters logs with facility auth and severity informational or higher. Log messages that pass the filter will be stored in the location `/var/log/remoteEvent`.

```
$RuleSet remoteEvent
auth.info       /var/log/remoteEvent

$InputRELPServerBindRuleset remoteEvent
$InputRELPServerRun 10514
```

It is possible to *bind a ruleset* which means that a specific input will pass its messages to a specific ruleset. The third line of the above example binds the rulset `remoteEvent` to a RELP listener and the fourth line activates the RELP listener to listen to port 10514. So when a message arrives at port 10514, it will be passed to ruleset `remoteEvent`. Rules from other rulesets are not relevant and will not be processed. This is very useful when, for example, local logs and remotly received logs are desired to be processed in differen ways.

A ruleset similar to the above example is used in both of the NCBs rsyslog configure files to filter log messages defined as security events in section 4.1. Besides facility auth, messages with the facility authpriv are also filtered. The location for storage of security events is the mount point of FSB which NCB considers as its own disk with NFS. Since the components have local disks in the lab system, there is a second rule in the ruleset that stores the same security events to the local disk of the ANCB. See Appendix A.1 for the configuration file used.

All machines in the system are configured to save all log messages to local disk and forward security events to the NCB VIP. Even the NCB1 and NCB2 store all their logs locally and forward the security events to the NCB VIP, and then the ANCB will take care of the messages received through the specific port where the listener is set up. We have three different listeners, one for each underlying transport protocol for syslog we tested; UDP, TCP and RELP.

To achieve reliable delivery of log messages, the log clients must know which messages have been received by the log server and which have not. Therefore the log clients need a reliable buffer to keep the created log messages if the server is unreachable. There are two relevant *queue modes* in rsyslog for the queue in interest, namely disk queue and in-memory queue. Queues in *disk mode* only use disk drives for buffering and do not buffer anything in memory. Disk queues are the most reliable but are also the slowest mode. An *In-memory queue* holds its elements in memory and are very fast, but the elements in the queue are lost if the system crashes. There is a submode of in-memory queue called *disk-assisted memory queue* (DA) which saves messages to disk if the memory is filled because of an unreachable log server. The DA queue can also be configured to save messages in memory to disk on a shutdown, though messages will still be lost at a crash. The DA mode has been tested in the lab system, but is not an option if true reliability is desired. Log clients using disk queues have mainly been tested in the test environment.

Authenticity and confidentiality in the transmission phase has not been the focus in this project, but rsyslog can be configured to encrypt syslog traffic with Transport Layer Security (TLS) [44]. By using TLS, the syslog messages are encrypted on the wire, and the sender knows who is receiving the syslog messages and the receiver knows who is sending them [11]. However this may cause problems since the system is a high availability system with redundant machines. There might be key exchange problems when the second machine becomes active. These problems are out of scope for this project.

5.3.2 Syslog-ng

Since several of the needed features in syslog-ng are provided only by the premium edition this is the one used in the lab system. The open source edition has not been tested. All machines run syslog-ng PE version 4.2.

Syslog-ng configurations are set up by four keywords; *Source,* which defines from where syslog messages should be received. *Destination,* defines to where and with which protocol syslog messages should be sent. *Filter,* sets up filtering rules for the sorting of logs. *Log,* combines a source with a destination and a filter.

Below is an example of a simple configuration file for a syslog server. The first line sets up a source that listens for incoming log messages sent with the RLTP protocol. The second line defines the destination file. The third line filters out messages that have the facility value auth and any severity value but debug. The last line defines a log path, which forwards all messages received from the source that match the filter to the destination.

```
source s_net { syslog(transport(rltp))};
destination d_file file{"var/log/remoteEvent"};
filter f_secEvent {facility(auth) and not level(debug); }
log { source(s_net); filter(f_secEvent);
destination(d_file); };
```

The configuration files for the two NCBs in the system are similar to this example. The filter rule is set up to filter out messages defined as security events. The configuration files in the lab system have three log paths, one similar to the example that receives messages from the net, match them with the filter and forwards them to a file on the NFS mounted directory at FSB. The second log path receives messages from the local system, matches them with the filter and forwards them to the virtual IP-address of NCB. The third log path stores all logs from local sources to the local disc. All other machines are configured to listen to local sources and forward the security events to the NCB VIP. The configuration files for the NCB and client can be found in Appendix A.2.

Beside RLTP one of the most significant improvements in the syslog-ng PE compared to the open source version is the reliable disk based buffering on client machines. If a client is unable to forward a message, for example due to a server or network crash, the client can store the message in a disk buffer and automatically resend the messages when the connection is reestablished. The buffer can be configured to be either normal or reliable. A normal disk buffer is faster but messages may be lost if the client crashes since the logs are stored in memory. If the reliable disk buffer is used, the messages are stored on the local hard disk and therefore persistent, no messages are lost even if the client syslog application is restarted [33].

Syslog-ng PE supports the TLS protocol for encryption and X.509 certificates for authentication of messages. This can be combined with the RLTP protocol to provide secure communication regarding reliability, confidentiality and authenticity. As for rsyslog this may cause problems due to the switching between redundant machines.

6 Execution of Test Cases

The test cases described in section 4.4 were performed by executing a C-program at the client machine including a syslog function, which creates a numbered syslog message and writes it to the Unix domain socket. Pseudocode for the C-program can be seen below. The facility and priority are set by the parameter values and are set to form a security event. The syslog daemon of the client handles the message and forwards it to ANCB. The program executes the syslog commands in a loop such that a syslog message is created once every second. This was considered to be sufficient since logs in the real SGSN-MME will normally be produced around that rate. For the most relevant test cases a transmission rate of 10 messages per second has also been tested. This was done to verify the conclusions from those test cases. This project investigated different protocols for reliable delivery during normal operation. A potential future work is to perform the tests with a higher transmission rate for preventing attacks like denial of service (DOS).

```
int nrOfmsg=100;
for (int lognr=0; lognr<nrOfmsg; lognr++)
    syslog(AUTHPRIV+INFO, "testMsg" + lognr);
    sleep(1);
```

After the client started to create syslog messages an event was initiated by the use of the *xm tool* which is a tool used to manage the DomUs from the command line in Dom0. Thus, starting, rebooting and shutdown of the two NCB machines, the two FSB machines and the client can be done from the Dom0 operating system. To initiate an NCB switchover the command `xm shutdown NCB1` was executed. The shutdown command allows NCB1 to perform an orderly shutdown. To inititate an NCB failover the command `xm destroy NCB1` was executed instead. This command instantly stops the operating system as would happen at a system crash or a power supply stop, which cause a risk for file system damage and data loss. FSB switchover and failover were similarly done with `xm shutdown FSB1` and `xm destroy FSB1`.

When the second machine had taken over all services and become the active one, the security event log file in FSB was verified to investigate how many of the sent log messages have reached the destination. Since the messages were numbered it was easy to identify if any of them were lost during transmission.

The client crash test was invented to test the buffer mechanism. The client syslog daemon was reconfigured to send the syslog messages directly to NCB1 instead of NCB VIP. After the client has started to create messages, NCB1 was stopped and shortly after the client was stopped. This way the client has some logs that have not been sent to the log server and the client machine should resend these once it is functioning properly again. The test includes all combinations of shutdown and destroy commands of NCB1 and the client.

The tests were done with various configurations regarding syslog transmission protocols and NFS transmission protocols. For rsyslog the tested log transmission protocols are UDP, TCP and RELP and for syslog-ng the tested protocols are UDP, TCP and RLTP. The protocol type was changed in the configuration file for rsyslog respectively syslog-ng at the client machine. The NCBs were configured to receive all different messages.

The FSB machines were configured to be NFS servers and the `/etc/exports` files in both machines were changed to include the directory where the security events should be stored. The options were set to be the same as in the real FSB boards in SGSN-MME to build the virtual lab system as near reality as possible. The NFS packets can be transmitted with either the TCP protocol or the UDP protocol. This is configured in the NFS clients. As stated in section 5.1, UDP is used as underlying transport for NFS in the real SGSN-MME because TCP decreases the performance greatly. However, TCP has also been evaluated in this thesis, since it provides retransmission and therefore more reliability than UDP.

7 Results and Discussion

This chapter presents the results of the test cases defined in section 4.4. It discusses different configurations for rsyslog, syslog-ng and NFS and how well these reach the goal of zero message loss. Section 7.2 discusses possible solutions to ensure integrity of the logged data in the storage phase.

7.1 Transmission Phase

Table 4 shows the number of lost log messages for different protocol configurations. The two last columns show the average value[8] of lost log messages from 10 tests. See Appendix B for a full documentation of the test results.

The results for the rsyslog implementation are very similar to the results for the syslog-ng for most of the test cases and therefore they will be discussed together.

Test case 1. The first test shows the number of lost messages during a log server switchover. NCB1 shuts down in the middle of the transmission and from that time ANCB can not receive log messages until NCB2 takes over the role as ANCB. Test 1a, 1b, 1d, and 1e show that using TCP for syslog transmission is slightly better than using UDP, regardless of which NFS transmission protocol that is used. As expected some logs are lost with TCP even though TCP provides retransmission. This verifies that TCP can only guarantee no packet loss as long as the network connection is kept as discussed in chapter 2. Test 1c and 1f show that RELP and RLTP solve this problem. As they also provide buffering queues and acknowledgements on the application layer, the goal of zero message loss was reached.

Test case 2. NCB failover was carried out in a similar manner as the switchover test. Test 2a and 2c show several more log messages were lost during a failover than during a switchover for UDP, TCP and RELP/RLTP protocols. Several of the log messages that were missing on the FSB disk existed at NCB1. This gave information that they have been received by ANCB but not stored to the FSB disk. This is caused by the NFS clients which buffer data in memory before writing it down to disk, consequently these messages will be lost at an ANCB crash.

Even though RELP/RLTP provide application based acknowledgments, this does not solve the problem. An acknowledgement is sent from the log server when the message is put in memory for processing, and not when it really is written to disk. The messages in memory will be lost during a failover and since the log client has received the acknowledgement the message will not be resent. Configuring rsyslog to acknowledge messages when it is really written to disk is not an available feature nor under development. Since rsyslog is open

[8] At a later stage worst case values may be more interesting. However, this is a pre-investigation and average values are used to do a comparison between the syslog applications.

#	Test	Log protocol	NFS	rsyslog	syslog-ng
1	NCB switchover				
1a		UDP	UDP	9	3
1b		TCP	UDP	1	1
1c		RELP/RLTP	UDP	0	0
1d		UDP	TCP	3	4
1e		TCP	TCP	4	0
1f		RELP/RLTP	TCP	0	0
2	NCB failover				
2a		RELP/RLTP	UDP	14	15
2b		RELP/RLTP	UDP (noac)	0	0
2c		RELP/RLTP	TCP	9	14
2d		RELP/RLTP	TCP (noac)	0	0
3	NCB failback				
3a		RELP/RLTP	UDP	13	1
3b		RELP/RLTP	UDP (noac)	0	0
3c		RELP/RLTP	TCP (noac)	0	0
4	FSB failover				
4a		RELP/RLTP	UDP	0	0
4b		RELP/RLTP	UDP (noac)	0	0
4c		RELP/RLTP	TCP	0	0
4d		RELP/RLTP	TCP (noac)	0	0
5	Client crash				
5a	NCB1 shutdown, client shutdown	RELP/RLTP	UDP (noac)	0	0
5b	NCB1 destroy, client shutdown	RELP/RLTP	UDP (noac)	128	0
5c	NCB1 shutdown, client destroy	RELP/RLTP	UDP (noac)	*	0
5d	NCB1 destroy, client destroy	RELP/RLTP	UDP (noac)	*	0
6	NCB failover & FSB failover				
6a		RELP/RLTP	UDP (noac)	0	0
7	FSB failover & NCB failover				
7a		RELP/RLTP	UDP (noac)	0	0

* Messages not sent before the client crash are lost. See discussion for test case 5.

Table 4: Test results for transmission phase. The last two columns state the mean values of lost logs sent with the transmission rate one log per second.

source one solution can be rewriting the code by oneself. However, providing acknowledgements later cause the negative effect that the client buffers can be filled up more quickly.

Another solution to the problem is to disable the NFS buffering on the log server so the log messages will be written to disk as fast as possible. Almost all NFS client operation need to know the file attribute for the file in interest and ask the server for it. The client caches this information for a period of time to reduce network and server load. NFS can be configured to use an option called *noac* which disables the file attribute caching and forces the client to check the server's version of the file attribute for each operation. This makes the writes to the server immediately visible to all clients but at the cost of more network operations.

By using the use of RELP/RLTP protocol for syslog transmission, a buffer mechanism for log messages at the client and the noac-option configuration of NFS, zero message loss is achieved during an NCB failover as seen at test 2b and 2d. However, the noac-option will probably decrease the performance in the system since it demands more network operations. Performance tests have not been performed and is a future work. Another potential future work is to find other ways to flush the buffer.

Even though this configuration seems to decrease the messages loss down to zero, the solution does not guarantee reliable delivery. The improvement is that the messages are forwarded to the correct place as fast as possible which increases the probability that they will arrive. Still acknowledgements are sent to the client before the message is written to stable memory so there is a possibility that they can be lost. A few tests for test 2b showed zero message loss even when they were sent with the increased transmission rate 10 logs per second.

Test case 3. Test case 3b and 3c shows that also during an NCB failback can zero message loss be achieved if RELP/RLTP is used as log transport protocol and NFS is configured with the noac-option. If the noac-option is disabled there is a problem that some log messages are overwritten by newer messages, so even if they have existed in the log file they are lost after a while. The problem is probably caused by the attribute caching and that file sharing does not work properly. It is very likely that this is only a problem in the virtual lab system as the SGSN-MME mounts the file system differently.

Test case 4. The FSB failover test is carried out in a similar manner as the NCB failover test. After FSB2 has taken the primary role, the log file is verified to discover the number of lost logs. If the RELP/RLTP protocol is used zero message loss is achieved regardless of NFS protocol and NFS options. As long as the log server runs normally, log messages are buffered on ANCB if the NFS server is unreachable. A few tests for test 4b showed zero message loss even when they were sent with the increased transmission rate 10 logs per second. Zero message loss is achieved also during switchover and failback.

Test case 5. The purpose of the client crash test was to investigate the reliable disk buffer queues on the client side in rsyslog and syslog-ng. The test was designed to let the client send its logs directly to NCB1 instead of the VIP of NCB. In the middle of transmission NCB1 is stopped and shortly after even the client is stopped. This way the client has some logs that has not been sent to the log server and the client machine should resend these once it is up again. The test has been performed with different combinations of shutdowns and crashes of the server and the client. This test is the only one that has shown significant differences between rsyslog and syslog-ng.

First a discussion of the results of rsyslog. If both the client and the server are stopped by a shutdown command no messages are lost neither with disk mode, shown by test 5a. If NCB1 is crashed but client stopped by a shutdown command, the results for test 5b showed a constant 128 message loss when using disk mode. Even if the transmission rate were changed the loss was still 128 log messages. This is probably caused by a whole block in memory not saved on disk.

The next two combinations, test 5c and 5d, both stop the client with a crash command. All messages that are created but not sent to the server are lost. However, by the use of disk mode the buffered messages can be found in a file in the work directory of rsyslog. Apparently they are not lost but the rsyslog daemon does not resend them when it is running again. Surprisingly, when new messages are created by the client, these will be put at the end of the queue and instead the first message in the queue will be sent. This seems like a very strange behavior and is probably caused by a bug in rsyslog.

Syslog-ng handles the client crash as expected regardless of how the machines are stopped. If NFS is configured with the noac-option, all created messages are stored in the reliable buffer and retransmitted later when the machines are running again.

Since the machines in the virtual lab systems have local disks, the buffering queues are stored there. However in the real SGSN-MME the local disks are partitions of the FSB disk and reached via the NFS protocol. Test case 2 has shown that reliability is affected by NFS caching. Therefore it is very likely that the messages generated by the log application but not yet stored to the NFS mounted local disk will be lost during a client crash. For that reason the test results from test case 5 are not fully trustworthy for the real SGSN-MME product.

Test case 6. This test performs an NCB failover and shortly thereafter also an FSB failover. Because the ANCB is the first machine to crash, this test shows a result similar to the single NCB failover. If RELP/RLTP is used for log transmission and the noac-option is used for NFS protocol then zero message loss is reached.

Test case 7. This test also shows results of a double machine crash. A failover of the FSB is followed by an NCB failover. For the transmission rate one message per second, the average loss is zero even though some of the tests resulted in one message loss by using RELP/RLTP and the noac-option. This is because when the FSB failover happens, the ANCB stores the messages in a buffer for later retransmission. When also the ANCB crashes these messages will be lost. The problem could be solved if the syslog daemon sends acknowledgements to the client when the message is written to disk.

Underlying transport protocols for NFS. In the SGSN-MME product UDP is used as underlying transport protocol for NFS because it provides better performance in the system. The test results in the virtual lab system shows that for reliably delivery of log messages the choice of NFS transport protocol has little importance. The results differ very little between UDP and TCP. The major issue regarding NFS for achieving zero message loss is the noac-option which disables file attribute caching and therefore makes the writes synchronous. However, as stated previously the noac-option only increases the probability that no mesasges will be lost but leaves no absolute guarantees that that will be the case.

7.2 Storage Phase

This section discusses several suggestions for secure log storage in Linux systems and how well they provide tamper resistance of the log files in the SGSN-MME. First is a discussion of some general and common techniques used for Linux systems, and then follows a discussion regarding hash chains and digital signatures and how these are used by the rsyslog and syslog-ng applications. Finally, the usage of a TPM is discussed.

The first and most obvious way to decrease the possibility for anyone to tamper with a file is to set the write permissions as restrictive as possible [35]. Only authorized users, as a suggestion only root users should have write permissions. Although this decreases the number of users that can tamper with the file, this is not a solution for the SGSN-MME, since several authorized users have root access to the system and are therefore allowed to change the file.

In a Linux system there is a possibility to set a so called append only flag to files. This makes it impossible to delete a file and writes are only allowed to the end of the file. To prevent an attacker with root access to unset the flag, one can change a startup script to remove the ability for changing the attribute. Still the attacker can change the script and then reboot the system [19].

An ideal solution that makes it impossible to change entries in the log file is to write all logs to a write once media, for example a DVD or forward them to a printer. This is physically impossible in SGSN-MME because too many log messages are generated [2].

Another method to protect the logs against malicious tampering is to forward the log entries to a remote log server. Then an attacker who has compromised the system also needs to compromise the remote system to manipulate the logs. Another possibility is that an attacker may disable the forwarding service but this will be logged before the service is disabled. The remote system needs to be well protected and only a restricted number of trusted people should have access [2]. This is a well known solution but since no such remote server exists for the SGSN-MME, this method can not be seen as an alternative.

In chapter 2 the scheme provided by Schneier and Kelsey was discussed. They propose a solution were the logs are stored locally and the remote machine is used only to validate the logs. This could be a solution if it exist a remote trusted machine which is not connected to the log server at all times.

Section 3.4 describes how cryptographic hash functions can be used to determine if the log data has changed. This method is used by rsyslog and syslog-ng to provide secure storage and seems to be a working solution. However in syslog-ng the log messages are collected to chunks and a hash value for every chunk is calculated from the log data in the chunk together with the hash value from the previous chunk. This forms a hash chain which makes it impossible to change data in the middle without destroying the chain. This is a good solution for integrity purpose but violates the reliability aspect of the system since the log messages waiting to form a chunk are stored in volatile memory and disappear[9] at a system crash [33].

An improvement of this would be to compute a cryptographic hash value for each entry individually. For example each entry may consist of a pair formed by the received log message and a cryptographic hash value of the previous entry. An attacker can manipulate messages in the middle but it will be detected since that will destroy the chain. A manipulation can only be done undetectably if the attacker rehashes the whole chain.

To prevent an attacker to rehash the whole chain all entries can be attached with digital signatures that guarantee the source of the messages. Each hash entry will be encrypted with the private key of the syslog application to form a signature. Both the hash value and the signature need to be stored in a secure location. Later the signature can be verified with the public key and the resulted hash value is compared to the stored hash value. If both this comparison as well as the hash chain is valid, integrity can be guaranteed.

This is a promising and well known solution to integrity problems. However there exist complications with this solution in the SGSN-MME. Where should the private key be stored? Several authorized users have root access and can use the private key to resign the messages after a modification. To solve this problem one need a trusted mechanism for storage of the private key.

One such trusted mechanism is the Trusted Platform Module (TPM), described in chapter 2, which can be used for secure storage of keys. A solution that makes use of a TPM to manage the hash chaining and signatures seems like the most promising solution for secure storage of log files in SGSN-MME.

[9]This has been verified in the lab system by a few tests that showed that several log messages were lost during a NCB failover even though the RLTP protocol and the noac-option for NFS were used.

8 Future Work

The results discussed in section 7.1 were only tested in a virtual imitation of the SGSN-MME. Before implementing centralized logging in the real product, tests similar to those described in section 4.4 needs to be performed in a real SGSN-MME. These test cases were designed to evaluate reliable delivery for the transmission phase, the storage phase has not been tested in this project and needs to be evaluated together with reliability since they may affect each other. We limited this project to reliable delivery in the transmission phase and integrity in the storage phase from the security aspects mentioned in section 4.1. The other security aspects for both transmission and storage phase might need to be considered as well if secure logging is desired.

We have mainly been focusing on the security aspects of centralized logging and did not consider performance of the system, a property which is very important in the real product. Performance tests need to be done before deciding which solution to use. It might be acceptable to lose some log messages if the performance is more important or the other way around. The noac-option in NFS might not decrease the performance at all or only by an acceptable amount, probably there are also other ways to flush the buffer in NFS than using the noac-option. A solution that decreases performance less than the noac-option might be found.

9 Conclusion

This thesis has shown that reliable transmission of log messages in SGSN-MME can be achieved. We performed seven different test cases and could show that in practice no log messages are lost in the virtual system with the right settings.

One required mechanism to achieve reliable transmission of syslog messages in SGSN-MME is the existence of a reliable buffer in all clients. The buffer stores log messages in hard disk until they are acknowledged by the log server, otherwise some security events may be lost if a client crashes while it still stores unsent log messages in the memory. This might be a problem in the real SGSN-MME since most of the boards do not have local disks. Storage to the reliable disk buffer is done by the NFS protocol, therefore it is very likely that the messages generated by the log application but not yet stored to the NFS mounted local disk will be lost during a client crash.

A transport protocol providing acknowledgments at application level is also required for achieving reliable transmission. Neither UDP nor TCP is sufficient since UDP does not offer any retransmission and TCP can not guarantee delivery if the connection to the server is lost. RELP and RLTP provide acknowledgments when the log server has put the received log messages in memory for the process. Even though this greatly increases the reliability it can result in message loss because some messages might be acknowledged but not written to disk. These will be lost during a log server failover as shown in our tests.

Another way to achieve zero message loss is to disable the NFS buffering. NFS clients normally cache the file attributes for some time to lower the number of network operations. When the noac-option for NFS is in use, the clients are forced to check the server's version of the file attribute for each operation which leads to synchronic writes. However, the noac-option might affect the performance of SGSN-MME.

The main issue regarding storage of log entries is that several users have root access. If secure storage should be provided by software in the system, the number of people having root access must be restricted to only include those that are authorized to access log files. Another and more promising solution is to use a TPM for storage of private keys.

Both rsyslog and syslog-ng PE gave very similar results of the different test cases for the transmission phase. Syslog-ng PE seems to be more stable since the client buffering does not work properly in rsyslog. However, rsyslog has the advantage of being open source software. The open source verion of syslog-ng does not offer reliable buffers or RLTP. Both rsyslog and syslog-ng PE offer all features needed for reliable centralized logging.

10 References

[1] U. E. Larson. *On Adapting Data Collection to Intrusion Detection.* PhD thesis, Chalmers University of Technology, 2009.

[2] S. Garfinkel, A. Schwartz, and G. Spafford. *Practical Unix & Internet Security.* O'Reilly, Sebastopol, CA, 3rd edition, 2003.

[3] B.A. Schroeder. On-line monitoring: a tutorial. *Computer*, 28(6):72 –78, June 1995.

[4] M.Z. Albari. *A Taxonomy of Runtime Software Monitoring Systems.* `http://www.informatik.uni-kiel.de/~wg/Lehre/Seminar-SS05/Mohamed_Ziad_Albari/vortrag-handout4.pdf`, 2005. [Online; accessed 2012-08-10].

[5] U. E. Larson, E. Jonsson, and S. Lindskog. A Structured Approach to Selecting Data Collection Mechanisms for Intrusion Detection. *Privacy, Intrusion Detection and Response: Technologies for Protecting Networks*, pages 1–39, 2012.

[6] K. Kent and M. Souppaya. *Guide to Computer Security Log Management.* NIST, September 2006. [Online; accessed 2012-09-10].

[7] M. Saxena, N. K. Singh, S. S. Thakur, and P. Kumar. A Review of Computer Forensic & Logging system. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(1), January 2012.

[8] N. Hall. *Creating a Secure Linux Logging System.* `http://www.sans.org/reading_room/whitepapers/logging/creating-secure-linux-logging-system_1529`, October 2004. [Online; accessed 2012-09-10].

[9] R. Accorsi. Log data as digital evidence: What secure logging protocols have to offer? In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 398–403, July 2009.

[10] R. Gerhards. *The Syslog Protocol.* `http://tools.ietf.org/rfc/rfc5424`, 2009. [Online; accessed 2012-04-25].

[11] *Rsyslog.* `http://www.rsyslog.com`, 2012. [Online; accessed 2012-04-25].

[12] *Syslog-ng.* `http://www.balabit.com/network-security/syslog-ng`, 2012. [Online; accessed 2012-04-25].

[13] *Systemd.* `https://fedoraproject.org/wiki/Systemd`, 2012. [Online; accessed 2012-04-25].

[14] L. Poettering. *Introducing the Journal.* http://0pointer.de/blog/projects/the-journal.html, 2011. [Online; accessed 2012-04-25].

[15] *Rsa Envision.* http://www.emc.com/security/rsa-envision.htm, 2012. [Online; accessed 2012-08-10].

[16] *Logrhythm.* http://logrhythm.com/, 2012. [Online; accessed 2012-08-10].

[17] *Arcsight.* http://www.arcsight.com/, 2012. [Online; accessed 2012-08-10].

[18] H. Tsunoda, T. Maruyama, K. Ohta, Y. Waizumi, G.M. Keeni, and Y. Nemoto. A prioritized retransmission mechanism for reliable and efficient delivery of syslog messages. In *Communication Networks and Services Research Conference, 2009. CNSR '09. Seventh Annual*, pages 158–165, May 2009.

[19] *How To Protect Your Logs from Tampering.* http://answers.oreilly.com/topic/424-how-to-protect-your-logs-from-tampering, 2009.

[20] *Techniques for ensuring verifiability of event log files.* http://security.stackexchange.com/questions/4320/techniques-for-ensuring-verifiability-of-event-log-files, 2012. [Online; accessed 2012-08-10].

[21] *Linked Timestamping.* https://secure.wikimedia.org/wikipedia/en/wiki/Linked_Timestamping, 2012. [Online; accessed 2012-08-10].

[22] *Write Once Read Many.* https://secure.wikimedia.org/wikipedia/en/wiki/Write_Once_Read_Many, 2012. [Online; accessed 2012-08-10].

[23] B. Schneier and J. Kelsey. Cryptographic support for secure logs on untrusted machines. In *The Seventh USENIX Security Symposium Proceedings*, pages 53 – 62, January 1998.

[24] Switzerland ISO, Geneva. ISO/IEC 11889 Information technology – Trusted Platform Module, 2009.

[25] Trusted Computing Group. *TPM Main Part 1 Design Principles*, 2011.

[26] R Accorsi. A secure log architecture to support remote auditing. *Mathematical and Computer Modelling*, 2012.

[27] Ericsson. *SGSN-MME 2011B Cp00*, 2011.

[28] F. Piedad and M. Hawkins. *High Availability: Design, Techniques and Processes.* Prentice Hall, Upper Saddle River, NJ, 2001.

[29] *What is HA?* `http://www.drbd.org/home/what-is-ha/`, 2011. [Online; accessed 2012-04-25].

[30] *Syslog.* `http://en.wikipedia.org/wiki/Syslog`, 2012. [Online; accessed 2012-08-29].

[31] C. Lonvick. *The BSD syslog Protocol.* `http://www.ietf.org/rfc/rfc3164.txt`, 2001. [Online; accessed 2012-04-25].

[32] W. Stalling. *Cryptography and Network Security.* Upper Saddle River, NY, 5th edition, 2011.

[33] BalaBit IT Security Ltd. *The syslog-ng Premium Edition 4 F2 Administrator Guide*, 2012.

[34] *What is Xen Hypervisor?* `http://www.xen.org/files/Marketing/WhatisXen.pdf`, 2011. [Online; accessed 2012-04-25].

[35] A. Silberschatz, P. Galvin, and G. Gagne. *Operating System Concepts.* Wiley, Hoboken, NJ, 8th edition, 2010.

[36] Sun Microsystems Inc. *NFS: Network File System Protocol Specification.* `http://www.ietf.org/rfc/rfc1094.txt`, 1989. [Online; accessed 2012-04-25].

[37] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. *Network File System (NFS) version 4 Protocol.* `http://www.ietf.org/rfc/rfc3530.txt`, 2003. [Online; accessed 2012-04-25].

[38] *Create High-Available NFS Server with Pacemaker.* `http://docs.homelinux.org/doku.php?id=create_high-available_nfs_server_with_pacemaker`, 2011. [Online; accessed 2012-04-25].

[39] *High Available NFS Server Using DRBD and Heartbeat on Debian 5.0 (Lenny).* `http://www.howtoforge.com/highly-available-nfs-server-using-drbd-and-heartbeat-on-debian-5.0-lenny`, 2010. [Online; accessed 2012-04-25].

[40] *DRBD Heartbeat and NFS on Debian.* `http://linux-ha.org/DRBD/NFS`, 2005. [Online; accessed 2012-04-25].

[41] F. Haas, P. Reisner, and L. Ellenberg. *The DRBD User's Guide.* `http://www.drbd.org/users-guide-8.3/users-guide.html`, 2011. [Online; accessed 2012-04-25].

[42] A. Beekhof. *Clusters from Scratch.* `http://www.clusterlabs.org/doc/Cluster_from_Scratch.pdf`, 2012. [Online; accessed 2012-04-25].

[43] R. Gerhards. *RELP - The Reliable Event Logging Protocol*. `http://www.librelp.com`, 2008. [Online; accessed 2012-04-25].

[44] F. Miao, Y. MA, and J. Salowey. *Transport Layer Security (TLS) Transport Mapping for Syslog*. `http://www.ietf.org/rfc/rfc5425.txt`, 2009. [Online; accessed 2012-04-25].

A Configuration Files

A.1 Rsyslog

Configuration file for the NCB machines.

```
### MODULES ###

$ModLoad immark # provides --MARK-- message capability
$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog # provides kernel logging support

# provides UDP syslog reception
$ModLoad imudp

# provides TCP syslog reception
$ModLoad imtcp

# RELP receive and send
$ModLoad imrelp
$ModLoad omrelp

### RULES ###

# Forward security events
$WorkDirectory    /root/rsyslogwork # location for spool files
$ActionQueueType Disk
# $ActionQueueType LinkedList # Disk Assisted
$ActionQueueFileName srvrfwd
$ActionQueueCheckpointInterval 1
$ActionResumeRetryCount -1 # infinite retries on insert failure
$ActionQueueSyncQueueFiles on # fsync
# $ActionQueueSaveOnShutdown on # save in-memory data on shutdown

auth,authpriv.info    :omrelp:169.254.0.21:20514 # RELP
# auth,authpriv.info    @@169.254.0.21:10514 # TCP
# auth,authpriv.info    @169.254.0.21:11514 # UDP

# Received security events
$RuleSet remote
auth,authpriv.info    /mnt/nfs/log/security # Store logs on FSB
auth,authpriv.info    /var/log/security # Store logs locally

# RELP
$InputRELPServerBindRuleset remote
$InputRELPServerRun 20514

# TCP
$InputTCPServerBindRuleset remote
$InputTCPServerRun 10514

# UDP
$InputUDPServerBindRuleset remote
$UDPServerRun 11514
```

Configuration file for the client machine.

```
### MODULES ###

$ModLoad immark # provides --MARK-- message capability
$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog # provides kernel logging support

# RELP send
$ModLoad omrelp

### RULES ###

# Store security events locally
auth,authpriv.info    /var/log/security

# Forward security events
$WorkDirectory    /root/rsyslogwork # location for spool files
$ActionQueueType Disk
# $ActionQueueType LinkedList # Disk Assisted
$ActionQueueFileName srvrfwd
$ActionQueueCheckpointInterval 1
$ActionResumeRetryCount -1 # infinite retries on insert failure
$ActionQueueSyncQueueFiles on # fsync
# $ActionQueueSaveOnShutdown on # save in-memory data on shutdown

auth,authpriv.info    :omrelp:169.254.0.21:20514 # RELP
# auth,authpriv.info    @@169.254.0.21:10514 # TCP
# auth,authpriv.info    @169.254.0.21:11514 # UDP
& ~
```

A.2 Syslog-ng

Configuration file for the NCB machines.

```
@version:  4.2
@include "scl.conf"
options { keep_hostname(yes); ts_format(iso); };
source s_local { internal(); system(); };
source s_net {
    udp();
    syslog(transport(tcp));
    syslog(
        port("4444")
        transport(rltp(tls_required(no)))
        ip-protocol(4) ); };
destination d_messages { file("/var/log/messages" fsync(yes)); };
destination d_secevent { file("/mnt/nfs/log/security" fsync(yes));
};
#defines a destination with transport protocol RLTP and a reliable
buffer
destination d_net {
    syslog ("169.254.0.3"
        port("4444")
        transport(rltp(tls_required(no)))
        ip-protocol(4)
        disk-buffer (
            mem-buf-size(10000)
            disk-buf-size(2000000)
            reliable(yes) ) ); };
#defines filter rules
filter f_secevent { facility(auth,authpriv) and not level(debug);
};
#send the received security events to destination defined as
d_secevent
log { source(s_net); filter(f_secevent); destination(d_secevent);
};
#sent own security events to destination defined as d_net
log { source(s_local); filter(f_secevent); destination(d_net); };
#save all own log messages at disk location defined as d_messages
log { source(s_local); destination(d_messages); };
```

Configuration file for the client machine.

```
@version:  4.2
@include "scl.conf"
options { keep_hostname(yes); ts_format(iso); }; source s_local
{ internal(); system(); }; #defines a destination with transport
protocol RLTP and a reliable buffer
destination d_net {
    syslog ("169.254.0.3"
        port("4444")
        transport(rltp(tls_required(no)))
        ip-protocol(4)
        disk-buffer (
            mem-buf-size(10000)
            disk-buf-size(2000000)
            reliable(yes) ) ); };


#defines a destination with transport protocol TCP
#destination d_net { syslog (“169.254.0.3” transport(tcp)); };


#defines a destination with transport protocol UDP
#destination d_net { udp (“169.254.0.3”);};


#defines destination file for log messages
destinations destination d_messages { file("/var/log/messages"); };


#defines filter rules
filter f_secevent { facility(auth,authpriv) and not level(debug);
};


#send the security events to destination defined as d_net
log { source(s_local); filter(f_secevent); destination(d_net); };


#save all log messages at disk location defined as d_messages
log { source(s_local); destination(d_messages); };
```

B Test Results

B.1 Rsyslog

#	Test	Log protocol	NFS	1	2	3	4	5	6	7	8	9	10	Average
1	NCB switchover													
1a		UDP	UDP	10	8	8	9	6	3	3	3	4	2	9
1b		TCP	UDP	4	4	4	4	4	4	4	4	4	4	4
1c		RELP	UDP	0	0	0	0	0	0	0	0	0	0	0
1d		UDP	TCP	4	4	4	2	4	4	3	3	3	3	3
1e		TCP	TCP	4	4	4	4	4	4	4	4	4	4	4
1f		RELP	TCP	0	0	0	0	0	0	0	0	0	0	0
2	NCB failover													
		UDP	UDP	29	38	15	17	18	32	25	14	29	33	25
2a		RELP	UDP	0	20	7	24	1	26	24	26	4	7	14
2b		RELP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
2b△		RELP	UDP (noac)	0	0	0	0	0	0					
		UDP	TCP	37	25	20	27	23	25	19	14	22	14	23
		TCP	TCP	17	22	14	16	13	13	23	16	19	19	17
2c		RELP	TCP	17	9	8	2	16	7	2	11	10	11	9
2d		RELP	TCP (noac)	0	0	0	0	0	0	0	0	0	0	0
3	NCB failback													
		UDP	UDP	9	4	2	18	23	23	9	25	6	20	14
		TCP	UDP	4	4	15	22	33	18	8				
3a		RELP	UDP	13	10	14	1	24	16	18	13	7	14	13
3b		RELP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
		UDP	TCP	16	31	22	19	5	5	28	3	15	31	18
		TCP	TCP	20	12	25	18	33	26	24	29	4	14	21
		RELP	TCP	0	0	0	0	0	0	0	0	0	0	0
3c		RELP	TCP (noac)	0	0	0	0	0	0	0	0	0	0	0

△ 10 messages per second was sent from the client to the server.

Table 5: Rsyslog, part 1.

#	Test	Log protocol	NFS	1	2	3	4	5	6	7	8	9	10	Average
4	FSB failover													
4a		RELP	UDP	0	0	0	0	0	0	0	0	0	0	0
4b		RELP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
4b△		RELP	UDP (noac)	0	0	0								
4c		RELP	TCP	0	0	0	0	0	0	0	0	0	0	0
4d		RELP	TCP (noac)	0	0	0	0	0	0	0	0	0	0	0
	FSB failback													
		RELP	UDP	0	0	0	0	0	0	0	0	0	0	0
5	Client crash	RELP	UDP	0	0	0	0	0	0	0	0	0	0	0
5a	NCB1 shutdown, client shutdown	RELP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
5b	NCB1 destroy, client shutdown	RELP	UDP (noac)	128	128	128	128	128	128	128	128	128	128	128
5b△	NCB1 destroy, client shutdown	RELP	UDP (noac)	128	128	128								
5c	NCB1 shutdown, client destroy	RELP	UDP (noac)	*	*	*	*	*	*	*	*	*	*	*
5d	NCB1 destroy, client destroy	RELP	UDP (noac)	*	*	*	*	*	*	*	*	*	*	*
6	NCB failover & FSB failover													
6a		RELP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
7	FSB failover & NCB failover													
7a		RELP	UDP (noac)	1	0	0	0	1	0	0	0	0	0	0

* Messages not sent before the client crash are lost.
△ 10 messages per second was sent from the client to the server.

Table 6: Rsyslog, part 2.

B.2 Syslog-ng

#	Test	Log protocol	NFS	1	2	3	4	5	6	7	8	9	10	Average
1	NCB switchover													
1a		UDP	UDP	8	3	3	3	1	2	3	3	2	4	3
1b		TCP	UDP	0	2	0	0	1	0	0	2	0	0	1
1c		RLTP	UDP	0	0	0	0	0	0	0	0	0	0	0
1d		UDP	TCP	4	2	2	9	5	3	2	8	2	2	4
1e		TCP	TCP	0	0	0	0	0	0	0	1	0	0	1
1f		RLTP	TCP	0	0	0	0	0	0	0	0	0	0	0
2	NCB failover													
		UDP	UDP	13	8	35	19	16	28	11	20	38	37	23
		TCP	UDP	23	30	18	24	16	24	14	28	17	19	21
2a		RLTP	UDP	5	21	19	6	7	12	13	19	18	30	15
2b		RLTP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
2b△		RLTP	UDP (noac)	0	0	0	0	0						
2c		RLTP	TCP	17	22	7	8	19	11	13	8	18	19	14
2d		RLTP	TCP (noac)	0	0	0	0	0	0	0	0	0	0	0
3	NCB failback													
		UDP	UDP	24	28	22	23	11	6	2	31	4	3	15
		TCP	UDP	4	5	7	7	2	4	4	4	5	2	4
3a		RLTP	UDP	4	0	0	0	0	0	5	0	0	0	1
3b		RLTP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
3c		RLTP	TCP (noac)	0	0	0	0	0	0	0	0	0	0	0
4	FSB failover													
4a		RLTP	UDP	0	0	0	0	0	0	0	0	0	0	0
4b		RLTP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
4b△		RLTP	UDP (noac)	0	0									
4c		RLTP	TCP	0	0	0	0	0	0	0	0	0	0	0
4d		RLTP	TCP (noac)	0	0	0	0	0	0	0	0	0	0	0
	FSB failback													
		RLTP	UDP	0	0	0	0	0	0	0	0	0	0	0
5	Client crash													
	NCB1 destroy, client destroy	RLTP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
6	NCB failover & FSB failover													
6a		RLTP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
7	FSB failover & NCB failover													
7a		RLTP	UDP (noac)	0	0	0	0	0	0	0	0	0	0	0
7a△		RLTP	UDP (noac)	4	3	7	5	5	6	7	5	5	6	5

△ 10 messages per second was sent from the client to the server.

Table 7: Syslog-ng PE.