TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology
Department of Software Science

ITC70LT
Sander Arnus 153191IVCM

PROVIDING RELIABLE LOG DELIVERY AND INTEGRITY OF LOGS

Master thesis

Risto Vaarandi
Ph.D
Senior Research Fellow

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Sander Arnus
May 17, 2017

Annotatsioon

Keskset logihaldust kasutades on oht, et sõnumid võivad kaduma minna. Teatud olukordades ei ole sellest probleemi, kuid logide digitaalse tõestusmaterjalina kasutamisel, ei saa lubada oluliste sõnumite puudumist. Sellest tulenevalt peab rakendama vahendeid, mis tagavad sõnumi tervikluse kontrolli ja võimaldavad seda teha ka kolmandal osapoolel, näiteks kohtul.

Käesolev lõputöö vaatleb logide keskse kogumise käigus tekkivaid olukordi, kus on oht sõnumed kaotada. Lisaks sõnumite transpordile vaadeldakse logide tervikluse tagamist logisid koguval seadmel. Lõutöös võrreldakse erinevaid tarkvarasid ja valitakse välja sellised, mis tulevad toime nii usaldusväärse logide edastamise kui ka logide tervikluse tagamise ja kontrolliga.

Kõigepealt vaadeldakse pakutavaid lahendusi üldisemalt. Esmase ülevaate põhjal valitakse välja nõuetele vastavad tarkvarad. Seejärel võrreldakse neid omavahel süviti. Valitud tarkvarasid testitakse autori poolt loodud keskkonnas. Testimise käigus kasutatakse esmalt vaikimisi konfiguratsiooni, et näidata kui palju võib sõnumeid kaotsi minna. Seejärel luuakse spetsiaalne konfiguratsioon ja viiakse testimine uuesti läbi. Testitakse kahe enimlevinud logi allikaga, nii failist kui ka süsteemi logist (/dev/log või systemd journal).

Lõputöö analüüsi põhjal valiti välja kolm syslog deemonit: rsyslog, syslog-ng Premium Edition ja Fluentd. Vaikimisi kaotasid kõik sõnumeid, kuid spetsiaalse konfiguratsiooniga ei kaotanud ükski nimetatud tarkvara failist pärinevaid logi sõnumeid. Kõige vähem sõnumikadu süsteemi logist sõnumite edastamisel esines Fluentd puhul.

Logide tervikluse tagamiseks sobib analüüsi põhjal kõige paremini Guardtime KSI. Antud lahendus pakub logi sõnumite signeerimist rea või bloki kaupa, rsyslog integratsiooni ning kohalikku ja kolmanda osapoole tervikluse verifitseerimist.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 91 leheküljel, viite peatükki, kolme joonist, kuute tabelit.

Abstract

There is a risk of message loss when using centralized logging. While this can be acceptable in certain situations, it is unacceptable when the log messages are to be used for digital evidence. In this case, measures have to be taken that provide integrity of logs and enable verification of integrity by third parties such as courts.

This thesis looks at message loss inducing situations when using centralized logging. In addition to covering reliable log message transportation, providing log integrity on a log collector is under discussion. Different solutions will be looked at and a selection is made based on how they manage reliable log delivery and integrity of logs.

First, available solutions will be looked at in general and initial selection is made based on requirements. An in-depth comparison follows with tests in a lab environment. A default configuration is used first to provide a baseline of how many messages can be lost without optimization. A custom configuration is then created and used to run the same tests a second time. Two of the most common log inputs, plain text files and system log source, are used as log message inputs.

Based on the analysis, three syslog daemons were chosen: rsyslog, syslog-ng Premium Edition and Fluentd. Using default configuration message loss occurred for all of the solutions. Custom configuration removed message loss for file log source. Fluentd had the lowest number of message loss when the custom configuration was used.

Based on the analysis, the best solution for providing log integrity is Guardtime's KSI. It provides capabilities to sign individual log messages or files, has rsyslog integration, and local and third party log integrity verification.

The thesis is in English and contains 91 pages of text, five chapters, three figures, and six tables.

Table of abbreviations and terms

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AWS	Amazon Web Services
BEEP	Blocks Extensible Exchange Protocol
BSD	Berkeley Software Distribution
BTA	Blockchain Timestamping Architecture
CE	Community Edition
CPU	Central Processing Unit
DB	Database
DNS	Domain Name System
EC2	Amazon Elastic Compute Cloud
EE	Enterprise Edition
FISMA	Federal Information Security Management Act
GB	Gigabyte
GLBA	Gramm-Leach-Bliley Act
GPRS	General Packet Radio Service
GSSAPI	Generic Security Services Application Program Interface
HDFS	Hadoop Distributed File System
HIPAA	Health Insurance Portability and Accountability Act
HMAC	Hash-based Message Authentication Code
HP	Hewlett-Packard
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Business Machines
JVM	Java Virtual Machine
KB	Kilobytes
KSI	Keyless Signature Infrastructure
MAC	Media Access Control
MSSQL	Microsoft SQL
NIST	National Institute of Standards and Technology

OS	Operating System
OSE	Open Source Edition
OSSIM	Open Source Security Information Management
OTS	OpenTimestamps
PCI DSS	Payment Card Industry Data Security Standard
PE	Premium Edition
PKI	Public Key Infrastructure
RELP	Reliable Event Logging Protocol
RHEL	Red Hat Enterprise Linux
RLTP	Reliable Log Transfer Protocol
RPM	Red Hat Package Manager
RSA	A security company - RSA Security LLC
SDK	Software Development Kit
SELinux	Security-Enhanced Linux
SGSN-MME	Serving GPRS Support Node – Mobility Management Entity
SIEM	Security and Information Event Management
SNMP	Simple Network Management Protocol
SOX	Sarbanes–Oxley Act
STOMP	Streaming Text Oriented Messaging Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UXP	Cybernetica's Unified eXchange Platform
VM	Virtual Machine
WAN	Wide Area Network

Table of contents

List of figures

List of tables

1. Introduction

Centralized logging is used by large organizations to comply with regulations and legislation. While providing compliance it can do much more, such as help investigate in near real time what is happening or what has happened in the past on systems or applications sending log messages. To take full advantage of this, components of centralized logging need to be configured accordingly. Using default configurations may work for a time but can introduce undesirable results like message loss.

What use is centralized logging when some of the log messages can go missing? While occasionally losing a number of log lines is not a problem to some applications, it can be an issue when a log line is needed as evidence. For example, when a log record about bank transaction is required but found to be missing. Worse still, what if someone has altered the log records after they were stored by log collector, be it intentionally or accidentally, and there is no way of knowing or proving if the log messages are accurate?

Using logs as evidence externally or internally requires that they are trusted. Log messages can be changed after being stored and if there is no method to detect modification then there is no way to trust the contents of log files. This issue is important for companies that have high value log sources with log messages which might need to be used as digital evidence (for example, bank transaction log records). In such a case guaranteed log delivery is needed with verifiable log integrity.

Although it is relatively easy to implement some sort of centralized logging, a simple syslog daemon restart can cause messages to be lost (not to be stored on central syslog server), when default or poor configuration is used. This loss is usually not noticed until the log in question is needed.

A number of syslog daemons are available and they are changing with time as new features are added and others improved. It is a time consuming task to go through all of them for companies that need to implement centralized logging with guaranteed delivery and integrity.

This thesis focuses on finding a way to deliver log messages from a number of log originators to a collector without losing any or as few messages as possible. In addition, it focuses on finding a solution for verifiable log integrity. The goal of this thesis is to conduct a high level analysis of available log delivery and log integrity solutions to find the most promising ones and analyze them in-depth. These best solutions will be benchmarked in

a test environment to measure and compare their effectiveness in practice. Scenarios that attempt to break log delivery will be tried out during testing. Testing is done in a separate lab environment with artificially generated test messages. Using a corporate environment with actual application and system log messages would provide more realistic results but unfortunately such testing was not feasible in the scope of this thesis.

Available solutions will be compared against requirements that medium to large size companies might have. Composed requirements are based on Swedbank AS, as the author is employed there, but they are general enough to be applicable to a wider range of organizations. Optimal configuration for best syslog clients and collectors will be written and tested.

As a constraint in this thesis, focus will be on solutions available for Linux operating systems. This focus is taken because Linux is a popular operating system on servers and also widely used in Swedbank that the requirements are based on.

Log delivery (from a log source to a collector) and log integrity are covered in the scope of centralized logging, meaning there is a large number of log sources and one or few log collectors. How logging is done from business application side is not covered here and it should be considered separately with specific applications in mind. In some situations it is needed that logging always happens and if it is not possible then the application is halted. Examples provided in this thesis do not stop business application when log collector is unavailable.

In scope of this thesis integrity of logs is provided after the log messages arrive to the collector. This focus is taken because of typical enterprise environment, where there is a large number of different systems. Installing and configuring integrity solution on each individual system would provide better grantees for integrity as it is closer to the source but it also has drawbacks: additional installed components (can affect the underlying system if they malfunction), keeping these components up to date, less efficient aggregation, more impact on network bandwidth (log messages and integrity proofs both need to be sent to log collector instead of just log messages). This thesis does not focus on attacks to log sources or log file tampering before they are sent to central log server. In addition, while confidentiality is important, it is not the focus of this thesis. All of the message transport protocols used in testing provide support for Transport Layer Security (TLS) and the transport channel can be encrypted if required.

Contribution of this thesis is a high level and in-depth analysis of current syslog daemons and integrity solutions. In-depth analysis includes detailed description of the solutions

and comparison of the features they offer. Custom configuration that focuses on reliable message delivery is created for rsyslog, syslog-ng Premium Edition and Fluentd.

The main contributions of this thesis are:

- Analysis and tests for current versions of syslog daemons and integrity solutions;
- Syslog daemon configurations that can be used as reference;
- Recommendations for choosing syslog daemon and integrity solution.

Chapter 2 describes what centralized logging is and consists of. It lists and covers the most popular syslog daemons and message transportation solutions. Previous work done on reliable log delivery is discussed in section 2.2. It is followed by what has been done in regards to providing log integrity.

Chapter 3 specifies requirements for log originators, collector and log integrity solution. After the requirements have been introduced a high-level overview explains why some solutions were selected and others not. Section 3.3 follows with an in-depth comparison of the selected solutions.

Chapter 4 covers how the lab environment was set up, test messages were formatted and generated, testing conducted, what test cases were used and how results were recorded. Section 4.3 provides a brief overview on how log integrity solutions work and how they could be integrated with syslog daemons. Section 4.4 gives a short overview of test results and provides recommendations based on example use cases.

The author would like to thank the supervisor of this work, Risto Vaarandi, for all the help, insight and ideas provided. Big thanks go to people from Stallion AS, who provided prompt assistance with acquiring syslog-ng Premium Edition for testing. Last but not least the author would like to thank his colleagues at Swedbank.

2. Existing solutions and related work

This chapter looks at existing work related to reliable log delivery and providing integrity of logs while using centralized logging. This thesis will mainly focus on solutions running on Linux as it is widely used as server operating system by businesses.

There are some papers which discuss guaranteed or reliable log delivery ([1], [2]), but they focus on systems with relatively small amount of generated messages. With small numbers of log messages it is possible to use methods like disk queues and sync after each message is sent. This approach provides the best reliability and makes it harder for messages to be lost. However, it is not usable with high log volumes, because of limitations set by disk input/output operations that become a bottleneck. Granted, with enough resources a storage solution could be built that permits this, but it is assumed that in most cases it will not be an option.

Some papers (like [1]) provide examples of old and legacy configuration for syslog daemons. Latest syslog daemon changes (new features and bug fixes) are not covered in past research. Several secure logging and log integrity schemes ([3], [4], [5], [6]) have been introduced over the years, but previous work does not describe how they compare and how can they be used in enterprise setting. This section attempts to answer aforementioned research questions.

In section 2.1 components of centralized logging will be discussed and focus of this thesis will be set. Most well-known syslog daemons will be described.

Section 2.2 covers what has been published in regards to reliable log delivery with the aim of identifying suitable solution to use and investigate further.

Section 2.3 defines some of the terms used in connection to providing integrity of logs and gives an overview of work done in that field.

2.1. Centralized logging

In this thesis, different terms relating to centralized logging and log management will be used. In order to provide clarity for the readers, some of them will be defined in the context of this thesis.

- Log file - is a file that consists of one or more log lines;
- Log line (record, event, message) - contains information about a single event, typically including a timestamp, process that generated the event and the event description itself;
- Logs - one or more log files.

Centralized logging is the process of sending system and application log messages from a number of independent systems to one centralized location where they can be processed, stored and analyzed.

The purpose for using centralized logging can vary. It can be used because of the benefits it brings or because of compulsion set by legislation and regulations. For example, to comply with Federal Information Security Management Act of 2002 (FISMA), the Health Insurance Portability and Accountability Act of 1996 (HIPAA), the Sarbanes-Oxley Act of 2002 (SOX), the Gramm-Leach-Bliley Act (GLBA), or the Payment Card Industry Data Security Standard (PCI DSS) [7].

PCI DSS (Payment Card Industry Data Security Standard) [8] is enforced on companies that store, process, and/or transmit cardholder data. One of its requirements is the tracking and monitoring of access to network resources and cardholder data using log messages.

Accorsi points out in [9] that log data is being increasingly used as digital evidence. Centralized logging could help manage all of the logs in an organization, provide evidentiary value, and certainty that the log records have not been tampered with.

Centralized logging provides a way to investigate in near real time what is going on or what has happened in the past on systems or applications sending log messages. Compared to local logging it offers the ability to see a big picture, where events can be correlated and user or application actions traced between systems. It is possible to use the collected log records to trigger actions, like running a script or triggering alert in case of a certain event. System and user statistics can be derived from log messages and reports generated. It could also help detect and defend against various security threats, for example DNS reflection attacks that Jose and A cover in [10].

In this and following sections the components and solutions used in centralized logging are considered. Definitions and terms from "The Syslog Protocol" RFC 5424 [11] are used in this thesis to refer to the components of centralized logging. Usually originator,

collector and analytics solutions are considered part of a centralized logging infrastructure (Figure 1). There can be relays between originators and collector or log messages can go directly to collector without going through relays. A log collector can do log analysis on the same machine or forward the log records to specific destinations.

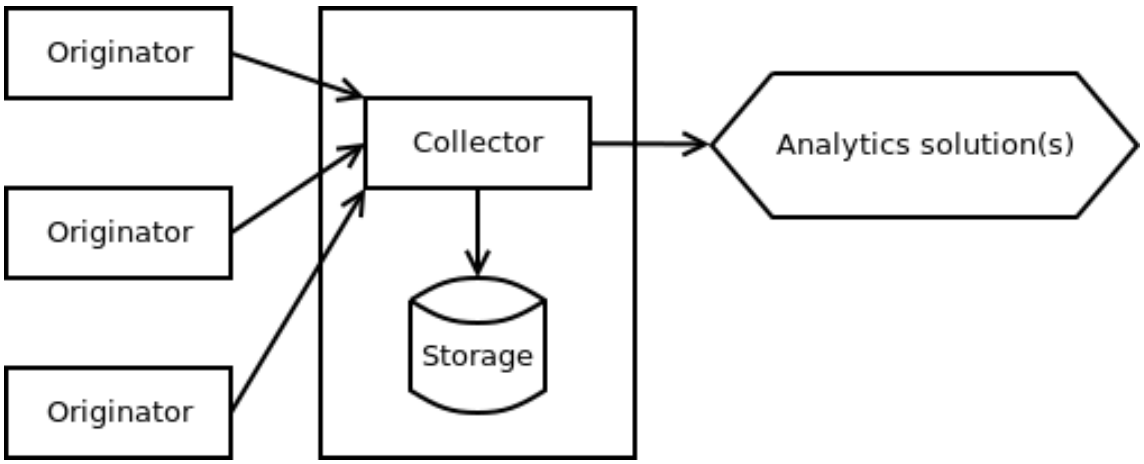

Figure 1. Centralized logging components

Originator

An originator (log source) is something that generates syslog content [11]. This can be a device, node or system that generates events containing information about its operation or information about applications running on the system. In this thesis, an originator is also referred to as a log source.

For an originator to be able to send log messages it needs a syslog daemon, a piece of software that can be configured to affect how system logging is conducted. There can be exceptions, like network devices that have a built in way of forwarding log messages with no specific syslog daemon. In addition, log messages can be sent using command line tools. For most cases, the assumption is made that a syslog daemon will be used and that it is possible to dictate logging behavior like which events are written to files, forwarded, normalised, filtered or dropped. For example, on most Linux machines the default syslog daemon is rsyslog [12]. There are other solutions available like syslog-ng [13], NXLog [14] and others that will be covered in sections 2.1.1 and 2.1.2.

Depending on the syslog daemon, the originator, relay, or collector can utilize different features like transport protocols, normalisation and filtering. These features will be looked at more closely and compared in chapter 3.

Relay
A syslog relay is responsible for accepting messages from originators or other relays and sending them to a collector or other relays [11]. Relays can have the same syslog daemon as an originator, although in this case the configuration and function is different.

This thesis does not focus on relays. They can be used in organizations as the first line in a centralized logging setup that execute the most computationally expensive actions like normalisation and filtering of incoming log messages before sending a refined flow of log messages to the collector. However, to keep testing simple and get more accurate results relays will not be included in the testing part of the thesis.

Collector
A collector has the task of collecting syslog content for further analysis [11]. A collector can do analysis on its own or forward log messages to separate analysis solutions. A collector also handles log storage. Logs can be stored locally on the collector, on a mounted Network File System or forwarded to dedicated storage systems.

It must be noted that collecting log messages is important but it would be pointless if the collected log records are not used. Log analysis is very important and, although focus is not on it in this thesis, it is a vital part of centralized logging.

2.1.1. Syslog daemons

Rsyslog
Rsyslog is a high-performance and feature rich log processing system [12]. It is open source but development and maintenance is mostly contributed by Adiscon [15] [16]. Adiscon also offers enterprise support for rsyslog. Rsyslog started as a regular UNIX syslog daemon, but has evolved into a system providing high-performance log processing, advanced security features, modular design with a high number of inputs, outputs and transforming options.

At the time of writing, the currently available stable version of rsyslog is 8.25.0. Rsyslog runs on Linux and can be used as a log originator, relay or collector. It can read log messages from a variety of local inputs like flat files, UNIX sockets, systemd journal but also receive messages from remote systems over Transmission Control Protocol (TCP), User Datagram Protocol (UDP) or Reliable Event Logging Protocol (RELP) [18]. Rsyslog can process log messages received from inputs using parser and message modification modules and output them using output modules [17]. RELP can provide reliable

log transfer, using application layer acknowledgment, between systems that have rsyslog or other solutions supporting RELP installed.

Rsyslog can also provide log integrity by specifying the use of Guardtime's Keyless Signature Infrastructure (KSI) in file output module [19].

MonitorWare Agent

Primary developer of rsyslog, Adiscon, also offers a commercial log agent for Windows platform called MonitorWare Agent [20]. This agent can be used on Windows hosts to act as log originator, relay or collector. It provides features like monitoring, listeners, data collection, alerting and more. At the time of writing currently available stable version is 11.0a build 481.

Syslog-ng

Syslog-ng comes in two forms. A free Open Source Edition (OSE) [13] maintained and developed by open source community and Balabit-Europe [21] and a commercial version called Premium Edition (PE) [22] also developed and maintained by Balabit-Europe [21]. For syslog-ng PE Balabit offers support directly if needed or through its reseller partners. At the time of writing, the currently available stable versions are 3.9 for OSE and 7.0.1 for PE (6.0.4 for latest Long Term Support version). In this thesis, focus will be on the Premium Edition as it has additional features that enable secure and reliable log transfer with Reliable Log Transfer Protocol (RLTP) [23].

Syslog-ng runs on Linux and PE offers Windows support. It can take on the role of a log originator, relay or collector. It provides features to securely and reliably transfer log messages, extract and process data, send to big data clusters, use queuing infrastructure, storing log messages in a database and more [24] [25].

Logstash

Logstash is a data collection system that can unify data from different sources and normalise it as needed by destination [26]. Logstash is open source and the company behind it is Elastic [27] who also offers enterprise support called subscriptions for Logstash and other products that go together with it (Elasticsearch, Kibana, Beats, X-Pack) [28].

At the time of writing currently available stable version is 5.3. Logstash is mostly used as a log collector or relay. Logstash runs on the Java Virtual Machine (JVM) and the system memory usage is higher than that of rsyslog or syslog-ng. Because of that it might not be possible or reasonable to run Logstash on log originators. Elastic has introduced lightweigth data shippers called Beats [29], that can be installed on hosts to have them

send log messages to Logstash. Logstash has a number of different inputs, filters and outputs that form a pipeline and enable to configure it as needed.

Reliable delivery can be achieved with the use of the lumberjack protocol [30] that provides application level message acknowledgment. Elastic's Filebeat client [31] can be used on log originator, that is installed on a host and sends log messages from files to Logstash via the lumberjack protocol. Logstash also support receiving events over RELP.

NXLog

NXLog, similarly to syslog-ng, has two editions available. One is open source NXLog Community Edition (CE) and the other is NXLog Enterprise Edition (EE) that contains additional features [14]. Both editions are available for Linux and Windows. NXLog, the company, also offers support for both syslog daemon versions.

NXLog has modular architecture. Only plugins that are actually needed can be loaded. Depending on the configuration it can act as a log originator, relay or collector. NXLog EE provides features for log integrity, like the ability to communicate with Timestamp Authority Servers and getting a timestamp on each log message. It is also possible to add a HMAC checksum to log messages to protect against tampering and forgery [32]. Reliable log delivery is provided using UDP or TCP, parallel processing, buffering, and I/O prioritization. Log message transport protocol using application layer acknowledgment is implemented using HTTP(S). However, NXLog CE does not have an HTTP input module.

At the time of writing, the currently available stable version is 2.9.1716 for NXLog CE and 3.0.1862 for NXLog EE.

2.1.2. Other solutions

Besides classical syslog daemons, there are other solutions that are capable of moving data between systems, storing and analyzing them. Log messages could also be moved using these solutions.

Apache Chukwa

Although Chukwa is not exactly a syslog daemon, it can be used to transport log messages in a reliable manner. Apache Chukwa is an open source data collection system with tools to display, monitor and analyze data. It is built on top of the Hadoop Distributed File System (HDFS) [33] and Map/Reduce framework [34]. At the time of writing currently

available stable version is 0.8.0.

Chukwa has agents that run on hosts that forward log messages to collectors. Requirement of having Hadoop up and running makes installing and collecting log messages more difficult than starting to use other syslog daemons. In addition, implementing log integrity proof generation can be more difficult with distributed file systems. Also, there seems to be no mention of enterprise support for Chukwa.

Scribe

Scribe was a project by Facebook that created a system for aggregating log data in real time from a large number of servers [35].

Zhang *et al.* compare Scribe against Chukwa in [36]. According to them Scribe makes significantly weaker delivery guarantees than Chukwa. Scribe server is able to do buffering and is responsible for the data after receiving it, however, because of that, failure of the Scribe server can cause data loss since originator does not keep a copy. Data can also be lost when the originator cannot contact working Scribe servers [36].

At the time of writing Scribe is no longer supported or updated by Facebook and with that the development of Scribe is at a standstill.

Fluentd

Fluentd is an open source data collector. It provides a unified data collection and consumption mechanism [37]. Fluentd can run as originator, relay or log collector. Creators of Fluentd have additionally created a more lightweight agent for data forwarding on Linux devices called Fluent Bit [38]. Fluentd project has been started and sponsored by Treasure Data, Inc. [39]. Treasure Data also provides support and stable distribution package (td-agent) of Fluentd. At the time of writing, the currently available stable Fluentd version is 0.12.31 (td-agent v2.3.4) and Fluent Bit version 0.10.

Architecture is similar to syslog-ng and NXLog where a pipeline is formed by using different plugins in sequence: input -> parser -> filter -> buffer -> output -> formatter. It has more than 500 plugins and supports features like memory- and file-based buffering.

Apache Flume

Flume can be used for collecting, aggregating and moving large volumes of log data in a distributed and reliable way. It has a flexible architecture with fault tolerance, reliability and recovery mechanisms [40].

Reliability of message delivery is achieved by keeping track of what was sent and received before deleting it on the sender. at the sender and receiver, messages are stored in a channel that can be disk or memory based. Flume has different sources, channels and sinks. Sources collect data and send it to channels for temporary storage where sinks read data and forward to destinations [41]. Flume can act as log originator, relay or collector.

Enterprise support for Flume is provided by Hortonworks [42]. At the time of writing currently available stable version of Flume is 1.7.0.

Apache Flume has been successfully used in a WAN deployed sensor network to send and aggregate data as outlined by Makeshwar *et al.* in [43]. It has also been used for data collection in anomaly detection setup described by Tangsatjatham and Nupairoj in [44].

2.1.3. Security and Information Event Management

Security and Information Event Management (SIEM) solutions are offered by several vendors. For example, HP [45], IBM [46], LogRhythm [47], McAfee (Intel Security)[48], RSA [49], SolarWinds [50] to name just some of them. They can contain a lot of functionality in a single appliance, like event correlation, log searching and visualization, report generation, analytics, notification, alerting and more. Unfortunately, SIEM solutions are expensive and it is not possible to test the full and actual functionality of these products without buying them. This is the main reason why SIEM solutions are not considered nor discussed in-depth in this thesis.

It has to be mentioned that there are open source SIEMs available. For example AlienVault OSSIM [51] offers a free solution in addition to its priced more feature-rich edition. However, for a medium or large enterprise the free solution is not enough because of the lack of scalability (single server only) and support.

SIEMs could be used in providing reliable delivery and integrity of log messages. They are mostly used as log collectors, but can also be used as analysis solutions that collectors forward log messages to. However, since they are mostly proprietary solutions requiring licenses, or limited functionality open source, then testing and reviewing them is not within the scope of this thesis.

2.2. Reliable log delivery

Reliable log delivery has been covered by Källqvist and Lam in [1]. They looked at it in the scope of Ericsson's SGSN-MME product. Their test results show that it is possible to achieve zero loss transmission for most of the test cases for both rsyslog and syslog-ng when disk queues are used. However, in case of large log volume that are expected in organizations using centralized logging it can be problematic or impossible to operate using disk queues and syncing every log messages to disk.

The now obsolete BSD syslog Protocol RFC 3164 [52] defines syslog to use UDP. UDP is known to be unreliable as it does not contain message reception acknowledgment like TCP does. Newer RFC 5424 [11] that obsoletes RFC 3164 does not specify transport layer protocol to use, but refers to RFC 5426 [53] that addresses transport of syslog messages over UDP. In RFC 5426 it is stated that UDP is an unreliable, low-overhead protocol and datagrams can be lost, messages corrupted or arrive out of sequence. It is recommended to use Transport Layer Security (TLS) specified in RFC 5425 [54]. According to [54], while TLS provides secure connection for syslog messages it does not provide fully reliable delivery because application layer acknowledgment is not used. TCP can provide protection against some forms of data loss, but when the connection is broken then sender cannot always know what was successfully delivered.

RFC 5848 [55] describes a mechanism called syslog-sign that adds origin authentication, message integrity, replay resistance, message sequencing, and detection of missing messages. This is accomplished by sending syslog messages with content called Signature Block that contains hashes of previously sent messages. According to [55] it is possible to verify if messages have been lost (not received by the collector) by reviewing the Signature Block information. However, it has to be noted that the implementation of RFC 5848 is not widely used.

Tsunoda *et al.* [2] point out why TCP is not a reliable delivery protocol for log messages and propose a better solution. Proposed solution uses UDP to transfer log records and application layer acknowledgment of message reception.

Other reliable delivery options that provide application layer acknowledgment but over TCP are rsyslog's RELP [18] and syslog-ng's RLTP [23].

RFC 3195 [56] also proposes reliable delivery of syslog by describing two mappings of the syslog protocol to TCP. It defines two Blocks Extensible Exchange Protocol (BEEP) profiles which are generic application protocol frameworks. They provide authentication, privacy and reliability [56]. Rsyslog has an input module that is capable of receiving log messages sent by this protocol. However, it is noted that there is no demand for it and if someone wants to put it into production then testing should be conducted [57].

Apache Chukwa [34] is offered as a reliable large-scale log collection solution in a paper by Rabkin and Katz [36]. In this paper the authors propose to use Chukwa as distributed log collection system. They look at two data delivery modes:

1. "Fast path" delivery that makes data available more promptly but does not provide reliability;
2. Reliable delivery that checks data after storage against local files on originator and resends if something is missing (introduces delay of several minutes onto data transfer).

Reliable delivery approach uses Chukwa agents on log originators that send log messages to Chukwa collectors, which then store them in HDFS. Requirements for this solution are Apache Hadoop [33] with HDFS storage layer, MapReduce computation model and Pig [59] data processing language. Although using Chukwa has a lot of benefits, it is dependent on Hadoop infrastructure.

Integrity checks and retransmission of missing log messages using Flume [40] is proposed by Tao, Yuan and Youxun in [58]. If missing data is found after log storage it gets resent to provide accurate log analysis.

David Lang constructed logging infrastructure that is able to process 100 000 log messages per second without much or any message loss [60]. In his approach he used Multicast MAC over UDP to send log messages to analytic engines. The need for this solution was introduced because of the network bandwidth limits in place at that time. It was not efficient to send multiple copies of log messages over the wire and sending to more than four destinations would have reached the network bandwidth limit. Instead, Lang came up with a solution to use Multicast MAC over UDP and send one copy to all analytic engines. Using UDP does not sound like the most reliable solution but according to the author [60], tests and practice of using it have shown it is possible to send several trillions of messages in bursts without message loss. In this case the tests were conducted in a local-area network under normal conditions (no interruptions) and focus was on sending

events from collector to several analytic solutions. Since UDP was used and there was no application layer acknowledgment messages would be lost in case of collector restart.

2.3. Integrity of logs

In this section different terms are used and to make it easier and clearly understandable some of the most frequently used terms will be defined below.

- Integrity of logs - is the accuracy and consistency of all the log records stored in a log file;
- Integrity proof - is information that can be generated at a specific time or in specific conditions. For example, when a log line is received or a log file is rotated. Integrity proof can then be used later to prove the log line (or file) has not been altered following the creation of integrity proof;
- Proof value of logs - is the trustworthiness of logs. This implies that log messages in a log file are intact and integrity can be proven using the integrity proof.

Application and operating system log messages are collected and analyzed in most large organizations for several reasons. They can provide early warning of systems impending failure, information to generate reports or for troubleshooting issues. However, log messages like any other digital information, can be modified or deleted thus rendering the evidence useless. An integrity proof of logs can be used to provide trust that the log messages are accurate and not modified.

An integrity proof is something that can be used to verify the integrity of logs or any other digital asset. It is a guarantee that the log messages are genuine and have not been tampered with. Thus, if the integrity proof is valid, the logs can be trusted to be an accurate reflection of their state at the time of recording. This is important when the need arises to provide trustworthy proof and log messages for use as evidence in the investigation of an issue. For example, imagine a bank needs to investigate the validity of a transaction. When log files record all transactions with necessary details and are trusted to be accurate then it is something to rely on for the investigation. Another example would be preventing system administrators from altering log files. Usually administrators have the highest privileges and can modify or delete files that others can not. However, when the integrity proofs of log files are generated, they cannot be manipulated or deleted without detection.

In Cybernetica's Unified eXchange Platform (UXP) [61] proof value of messages is achieved by cryptographically signing and time-stamping them. This is a common method: using some form of timestamping in conjunction with a cryptographic operation to fix the state of a message to a point in time.

Accorsi and Holt have discussed proof value of logs in a number of papers ([62], [63], [64]). They use different methods in generating integrity proofs. For example, some of the components like message authentication codes, secret values between two devices and a chain of hashes are used.

The problem of checking log integrity for forensic investigation is identified by Zhang *et al.* in [65] and a solution is proposed. The solution uses a hashing schema with Shifted Transversal Design Group Testing algorithm to calculate hash values for all records in a log file as the integrity proof and precisely locate the records which have been corrupted.

Several schemes have been proposed that aim to grant log integrity. For example, by Bellare and Yee [66] who use forward integrity message authentication scheme or Schneier and Kelsey [67] with secret authentication key, trusted verification machine, per-record encryption keys and permission masks. However, they both have weaknesses of storage inefficiency, vulnerabilities to delayed detection attack and truncation attack, identified by Ma and Tsudik in [4]. Over the years improvements have been proposed: Schneier and Kelsey [3] (relies on a trusted third party), Holt [68] (storage inefficiency, vulnerable to truncation attacks), Stathopoulos, Kotzanikolaou and Magkos [69] (relies on trusted third party).

RFC 5848 [55] addresses log integrity in addition to origin authentication, replay resistance, message sequencing and detection of missing messages. However, this mechanism is not widely used.

A log signing scheme is proposed by Buldas, Truu, Laanoja and Gerhards in [6] that enables verification of log integrity and presentation of log record with a compact integrity proof without revealing information about other log messages in a log file. This is used by the Guardtime's KSI that is discussed below.

One option would be to use blockchain based solutions. Blockchain is clearly defined by Back *et al.* in [70] as: "a well-ordered collection of blocks, on which all users must (eventually) come to consensus. This determines the history of asset control and provides a computationally unforgeable time ordering for transactions". If the integrity proof of a log file would be recorded in a blockchain then it is possible to refer back to the blockchain to verify if the log file has been changed after publishing the integrity proof.

Possibilities exist to use permissioned blockchains, such as Guartime's KSI [71]. This solution can be used with the popular Linux syslog agent, rsyslog [12], which makes it easy to adopt onto Linux based environments. KSI is described by Buldas, Kroonmaa and Laanoja in [72] as a service that can be used for time-stamping and digital signatures. What keyless means in this context is that verification of signatures can be done without relying on secret keys, thus the solution is not vulnerable to key compromise. Signing of documents or log messages and files is done by hashing them. This hash is then aggregated to form a hash tree and the root hash value (top hash that is the result of concatenation of hash values of all child nodes) is used in publishing a value that can be used during verification. Compared to previous solutions, KSI improves on the computational cost, storage size of generated proof and eliminating key compromise. One thing to remember is that Guartime's KSI is a service, meaning a fee has to be paid for using it. In this thesis KSI SDK integration model is discussed.

A proposal is presented by Cucurull and Puiggal in [5] to use the Bitcoin blockchain to hold integrity proofs of logs. It is possible to use special type of output that enables the inclusion of up to 75 bytes of information to transactions that are published in the blockchain [5]. Using Bitcoin's blockchain is not free and it has constraints like size of block (limits number of transactions) and the time it takes for blocks to be added to the blockchain [5].

OpenTimestamps [73], created by Peter Todd, is a relatively new solution using blockchain technology to provide proof of data existence at a specific time by using hashing and timestamps. It currently only uses the Bitcoin blockchain, but the developer has expressed his desire to add others in the future. OpenTimestamps provides two ways of timestamping data. One takes longer by waiting for confirmation from the Bitcoin blockchain (from around 10 minutes to several hours). The other uses public calendar servers, hosted by OpenTimestamps, and produces "incomplete" timestamps in roughly one second that relies on the calendar servers for verification. Calendar servers then interact with the Bitcoin blockchain and store the proof indefinitely for everyone to see and use for verifications. It is possible to later upgrade "incomplete" timestamps to include all the relevant proof information from the Bitcoin blockchain. While it is promising, OpenTimestamps is not

yet mature for production implementation as it is in alpha and lacks enterprise support.

Stampery is another relatively new company that uses public blockchains to provide proof of ownership, existence, integrity and reception of data [74]. They have integration with Estonian e-Residency and provide API for integrating other applications with their Blockchain Timestamping Architecture (BTA) [75]. BTA uses the Bitcoin and Etherium blockchains and allows anchoring of a virtually unlimited amount of data by using hash aggregation with hash trees. Stampery also has a solution, currently in pre-alpha, called Trailbot [76]. Trailbot enables monitoring of file changes, including changes in log files. It offers several Smart Policies like file rollback, backup of files, embeding file hash to blockchains, Zapier integration, email notifications or shutting down of an entire system. Trailbot requires three separate components: a watcher, a client, and a vault. A watcher is installed on servers that need monitoring and a client is installed on a computer that manages watchers, defines policies and views events. A vault is a backend component that can be set up separately or a public server offered by Stampery can be used. The currently available developer preview is free of charge but not yet ready for production use.

3. Analysis

The analysis part of this thesis defines requirements for the components of the log gathering and integrity solution. Requirements are based on the environment of Swedbank AS [77]. The logging environment of Swedbank, and requirements, are comparable to typical corporate environment. Because of that, the results of the analysis are applicable to not only one company but a wider range of organizations.

This chapter gives a brief overview of different possible components, explaining why some were chosen to be compared more thoroughly while others were not. Finally, based on the compliance to requirements a number of suitable applications will be selected for testing in lab environment.

Figure 2 illustrates the parts that are in focus in this thesis. Originators have syslog daemons installed and they forward log messages to the collector. Collector also has syslog daemon that is configured to listen for incoming log messages, process and store them. Depending on the integrity solution the implementation of log processing can be different. Integrity solution could be part of syslog daemon code where incoming log lines are immediately forwarded to be processed by the integrity solution. Also, it could be separate, where the integrity solution is instructed to process specific log files in a periodic fashion, like every 10 minutes or after file rotation.

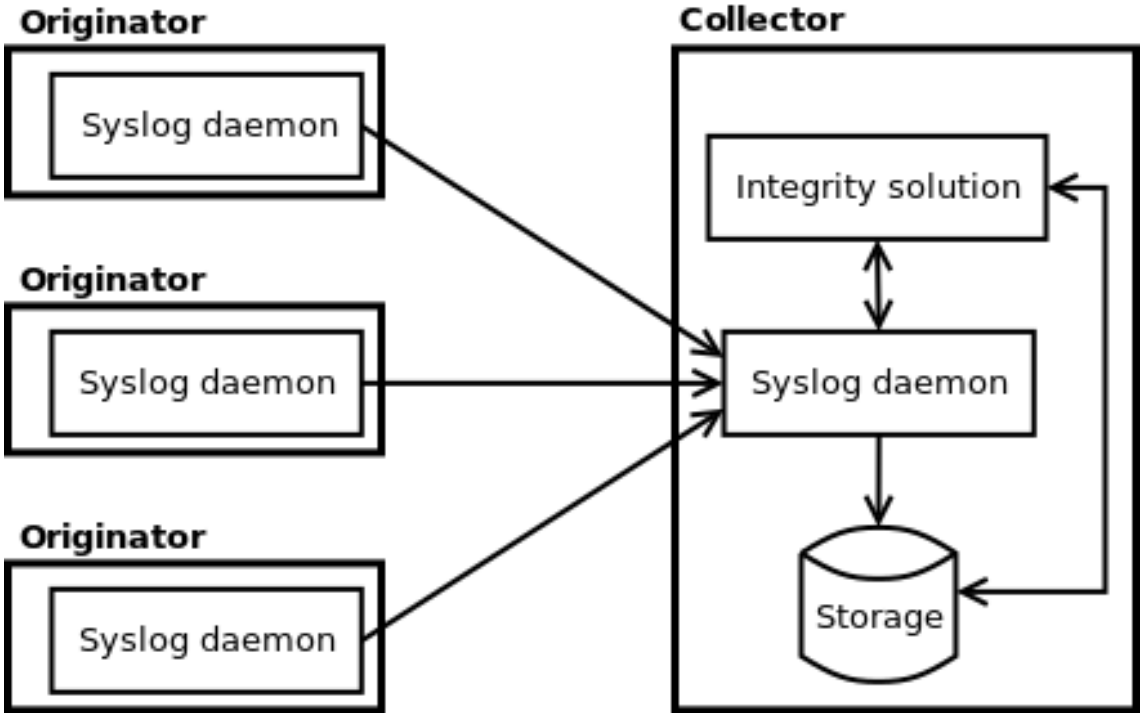

Figure 2. Centralized logging components in focus

3.1. Solution requirements

Requirements are described in Tables 1-3. The first column is the number of requirement to make it referable in text. The second column is the requirement description and the third column is an importance category varying from high (mandatory) to medium (recommended) to low (nice to have). Each table is followed by a detailed explanation of the proposed requirements.

Requirements are divided into two sections. First, in section 3.1.1, it is looked at what is required from the syslog solutions. This includes the log sender and collector. Requirements for the log sender and collector are separated due to the different function they have. Secondly, requirements for the log integrity solution in section 3.1.2 are listed.

3.1.1. Syslog daemon requirements

Originator

Table 1. Syslog originator requirements

Nr	Requirement	Importance
1	Works on Linux operating systems, specifically CentOS 7 and Red Hat Enterprise Linux 7	High
2	Has enterprise support available	High
3	Supports reading log messages from Systemd Journal	High
4	Supports reading log messages from Linux log socket	High
5	Supports reading log messages from files	High
6	Supports the use of wild-cards (*) when reading from files instead of specifying file names. Example, /var/log/*.log, for specifying all files in the /var/log directory that end with .log extension	High
7	Supports reliable log delivery with application layer acknowledgment over a network	High
8	Supports the use of memory buffers	High
9	Supports the use of disk buffers	High
10	Supports filtering of log messages based on syslog facility and severity	High
11	Supports filtering of log messages based on message content	High

Continues...

Table 1 – *Continues...*

Nr	Requirement	Importance
12	Supports applying specific formats/templates to log messages	High
13	Supports adding additional data (metadata) to log messages	High
14	Regularly updated and patched by the developer	High
15	Supports encryption of network transport channel (TLS)	High
16	Supports the use of TCP to send log messages over a network	Medium
17	Supports the use of UDP to send log messages over a network	Medium
18	Documentation of each feature is available	Medium
19	Supports generation of syslog daemons statistics, like number of log messages processed per input, number of log messages forwarded per destination, system resources used	Low
20	New functionality can be added to the software, either ordered from the vendor or developed in-house (open-source)	Low
21	Works on Windows Server 2008 (R2)	Low
22	Works on Windows Server 2012 (R2)	Low
23	Works on Windows Server 2016	Low

Table 1 lists the requirements for syslog daemon on originator and the importance of the requirements. As Linux servers are widely used in enterprises [78], the requirement for syslog daemon to work on them is an important one. Red Hat Enterprise Linux is used by many companies and that is why it is specifically mentioned, along with CentOS that is a free alternative.

The solution also needs to have professional support available to help discover, troubleshoot and fix issues. Support can be provided directly by the vendor or by a competent third party who has in-depth knowledge of the software.

Requirements number three to six cover most frequent log inputs. Requirements seven and 16-17 list different log transport protocols that need to be supported. Encryption of the transport channel should be possible using TLS, requirement 15.

Log buffering to memory is needed in case the collector is unavailable and log messages are read from log socket. Buffering to disk is needed when the originator shuts down while in the middle of log forwarding. It should be possible to write the location of where processing was stopped to disk and continue from there after start-up.

Support for filtering of log messages is required to define, based on the content or priority, how log messages are treated. For example, it might not be needed to forward log messages with debug or info severity but everything else needs to be sent. Using filtering it is possible to save network bandwidth and resource usage on the collector by filtering out nonessential log messages.

Modifying the format of log messages is also important as some applications do not follow syslog standards for logging and the events need to be rewritten to follow uniform rules and make the log messages understandable to analytic tools.

Without documentation or with limited and poor documentation, implementing and maintaining the solution will be close to impossible. Available features with description, configuration syntax, troubleshooting and examples should be documented.

The syslog daemon should also be actively improved and patched by the developer. Without patches the security of the whole system could be affected by running software with vulnerabilities.

Option to gather statistics on how the syslog daemon behaves would be valuable. Statistics like where most of the log messages originate (log socket or file) and if some events are dropped because of rate limiting would help fine-tune originators and the whole centralized logging system.

Using one product for long term can introduce new use cases that were not thought of in the beginning. For that the ability to add new functionality can prove useful. Open-source solutions will be favoured as it allows for in-house development. Ordering development from a vendor is also an option.

Depending on the organization, the ratio of Linux to Windows servers can vary. In Windows centric environments, originators run on Windows operating system and thus need a syslog daemon that supports it. Nevertheless, in this thesis focus is primarily on solutions made for Linux.

Collector

Table 2. Syslog collector requirements

Nr	Requirement	Importance
1	Works on Linux operating systems, specifically CentOS 7 and Red Hat Enterprise Linux 7	High
2	Has enterprise support available	High
3	Supports reception of log messages with application layer acknowledgment over a network	High
4	Supports the use of memory buffers	High
5	Supports the use of disk buffers	High
6	Supports filtering of log messages based on syslog facility and severity	High
7	Supports filtering of log messages based on message content	High
8	Supports writing log messages to files	High
9	Supports applying specific formats/templates to log messages	High
10	Regularly updated and patched by the developer	High
11	Solution is scalable	High
12	Supports encryption of network transport channel (TLS)	High
13	Supports the use of TCP to receive log messages over a network	Medium
14	Supports the use of UDP to receive log messages over a network	Medium
15	Supports generation of syslog daemons statistics, like number of log messages processed per input, number of log messages forwarded per destination, system resources used	Medium
16	New functionality can be added to the software, either ordered from the vendor or developed in-house (open-source)	Medium
17	Documentation of each feature is available	Medium

Table 2 lists the requirements for syslog daemon on log collector. Most of the requirements are the same or similar as for the log originator covered below Table 1 and they will not be discussed again. Instead of sending log messages, except forwarding to analytic solutions, a collector receives, processes and stores them.

For log collectors the ability to scale is important, while it can be done horizontally, by adding more collectors, it should be doable vertically as well. Making more CPU cores available, increasing CPU clock speeds, amount of memory, network bandwidth or storage performance should translate into increased number of log messages being processed

per time unit. Need for scaling can be identified from performance data (requirement 15) as it should provide relevant statistics to identify bottlenecks.

3.1.2. Log integrity

Table 3. Log integrity solution requirements

Nr	Requirement	Importance
1	Works on Linux operating systems, specifically CentOS 7 and Red Hat Enterprise Linux 7	High
2	Has enterprise support available	High
3	Local verification of log integrity	High
4	Public verification of log integrity	High
5	Solution is scalable	High
6	Can generate integrity proof per file	High
7	Efficient integrity proof generation and verification (time and size)	High
8	Can generate integrity proof per log line	Medium
9	Has documentation available	Medium
10	File ownership can be proved	Low

A log integrity solution, like syslog daemons, should work on Linux operating systems and have technical support available from the vendor. This support is needed during implementing, maintaining and troubleshooting.

Different verification means are needed for daily operations and special cases, for example, providing log messages as digital evidence. The solution should provide internal verification where employees can verify log integrity, and public verification where log file (and integrity proof if needed) are shared with a third party outside of the company.

Increasing the number of incoming log messages can increase resource demand on integrity solutions. Because of that, both vertical and horizontal scalability are desirable features.

Options to generate integrity proofs per file or per log line should be available. It is then possible to define how often an integrity proof is generated (after how many log lines) based on the needs of the application or platform owner sending log messages.

Generating and verifying integrity proofs should be as efficient as possible. The size of and time required to generate the proof should be as small as possible.

After generating an integrity proof it would be beneficial to also have the possibility to prove who the owner of the data was. Proof of ownership should provide assurance that what company A presents as digital evidence was actually signed by company A and did not originate from a third place.

3.2. High-level overview of available solutions

In this section syslog daemons covered in sections 2.1.1 and 2.1.2 are looked at. Also, integrity solutions from section 2.3 are reviewed. Although, the solutions in section 2.1.2 are not exactly syslog daemons, they can be used for transporting log messages. To keep things simple from here on they will be referred to as syslog daemons. Filtering of the solutions will be made based on how they comply to requirements and three most compliant solutions will be selected. Short explanations why something was chosen or not will be provided. In section 3.3 detailed comparison of the selected three solutions will be conducted. Integrity solutions will be compared in a similar manner.

3.2.1. Syslog solutions

Rsyslog
Rsyslog covers all the requirements set for both log originator and collector, with the exception of running on Windows platforms. However, as Windows support is a low priority requirement this does not stop us from selecting rsyslog as one of the candidates and looking at it in-depth.

MonitorWare Agent
MonitorWare Agent runs on Windows platforms and covers the requirements not met by rsyslog. Despite satisfying these low level requirements it does not satisfy high level requirement of running on Linux operating systems and because of that is not considered as a candidate.

Syslog-ng

Syslog-ng, similarly to rsyslog, covers all of the requirements set. In addition, syslog-ng Premium Edition can work on Windows platforms. Based on that syslog-ng will be selected as the second candidate to be compared and tested.

Logstash

Logstash meets all the requirements for both originator and collector with some shortcomings in buffering. The size of in-memory queues(buffers) is fixed and not configurable. Persistent queues (disk buffers) have been introduced recently. Using them enables writing the queues to disk and protects against data loss. However, this feature is in beta and a warning is given on the website about deploying to production at your own risk [79]. Logstash runs on JVM and it consumes considerably more memory compared to rsyslog or syslog-ng. Taking that into consideration, it might not always be possible to run Logstash as originator. Elastic, maintainer of Logstash, has addressed this by creating Beats. In regards to reliable log delivery, Logstash can use either Lumberjack, between Beats and Logstash, or RELP, to receive log messages from rsyslog. However, RELP plugin is community maintained and might not right away support latest fixes and improvements. Due to the high resource usage and limited queue configuration, Logstash is not chosen to be compared further.

Elastic Beats

Beats [29] are lightweight data shippers that can be used on log sources. There are different Beats each with very specific purpose, like Filebeat that reads from files and is available on Linux, Windows and Mac platforms. There is a high number of community beats available, like Twitterbeat, Pingbeat, Nginxbeat, Httpbeat or Journalbeat [80]. However, they can lack documentation, example is Journalbeat [81], and being community driven it might take time to get support when needed. There also seems to be no Beat for reading directly from Linux log socket (/dev/log), which means another syslog daemon needs to run and write log messages to a file from where Filebeats can read. This seems unreasonable if the alternative is to do everything with one syslog daemon. Because of the lack of documentation and features (reading from Linux log socket) Beats is not considered as an option.

NXLog

NXLog covers most of the requirements set for originator and collector. It also supports running on Windows platform. Application level acknowledgment is handled with HTTP(S). NXLog Community Edition (CE) has output module, om_http, that can be used to send messages over HTTP(S) protocol providing HTTP-level acknowledgment, but this is not supported as input by most syslog daemons. NXLog Enterprise Edition (EE) has

both an output and input module for sending and receiving messages using HTTP(S). NXLog EE has a module that can send hashes of log messages to Time-Stamp Authority server and get back a cryptographic Time-Stamp signature that proves message existed at the time of signing and can be used for verification later [82]. However, it comes with a noticeable performance impact when large numbers of messages need to be timestamped, since each message generates an HTTP request. Processing a batch of messages is not currently possible. Also, NXLog documentation does not indicate that an input for systemd journal exists [82]. Support for systemd journal was one of the high importance requirements. Due to no systemd journal support NXLog will not be covered further is this thesis.

Apache Chukwa

Chukwa can be used to transport log messages, but it will not be considered an option in this thesis because of its requirement on having additional infrastructure set up (Hadoop).

Scribe

Scribe has not been actively maintained or developed for a time and because of that it will not be consider as an option.

Fluentd

Fluentd has a huge number of input, output, filter, parser and formatter plugins created by the community and inventors of Fluentd. Due to the large number of plugins some of them can be broken, not updated for a long time or have poor documentation. There is a "Certified" status that is granted to plugins developed by core committers or committing companies. Plugins needed to cover our requirements are documented, however, some of them are not certified. Similarly to rsyslog, Fluentd does not currently run on Windows platform. Nevertheless, it covers all other requirements and is selected as a candidate for originator and collector to be tested further.

Fluent Bit

Created by the inventors of Fluentd, Fluent Bit could be considered as a more lightweight agent on originator. Due to the lack of inputs for systemd journal and Linux log socket, it will not be tested further. For system and process health monitoring Fluent Bit seems to offer excellent support by having CPU, memory and kernel log inputs and outputs.

Apache Flume

Flume does not have an input for reading Linux log socket or systemd journal, because of that it cannot be use on the originator. In addition, similarly to Logstash, Flume requires Java and runs on JVM, meaning the system footprint is larger than other syslog daemons.

Flume uses transactional approach between its components to guarantee log delivery. This covers the requirement for reliable transfer between originator and collector if both are running Flume. However, since it cannot be used on the originator and transactional approach is lost, there is no benefit in choosing to run it as collector.

3.2.2. Integrity solutions

There are very few solutions that offer third-party verifiable log integrity with enterprise support. Due to the third party verifiability requirement and lack of enterprise supports most log integrity schemes discussed under section 2.3 are not feasible options, leaving little to compare. To the best of our knowledge, at the time of this writing, there is only one suitable option: Guardtime's KSI. Since the introduction of blockchain, the number of companies that attempt to utilize blockchain is increasing. Some of them use it to provide provable log and data integrity. However, most of the offered solutions are not mature (in alpha or beta phase) and require more work before they are suitable for larger organizations. Along with Guaradtime's KSI, which is the only solution compliant with all the requirements, two other examples are looked at below.

Guardtime KSI
Guardtime's Keyless Signature Infrastructure is the only solution currently available that covers the entire requirements set. It uses hash-function cryptography to generate and verify integrity proofs for logs and other data. In the case of KSI, signatures are created that prove the owner of logs in addition to the integrity. KSI uses blockchain to store the integrity proofs in an append only way that can later be referred to for verification internally or by a third party. The KSI blockchain is permissioned, writing access to it is controlled, consensus achieved within one second, grows linearly with time and is scalable. The solution is not directly affected by an increase in number of log messages that need to be signed as hashes of log messages are used and aggregated to hash tree. The root hash of the tree is then used in the generation of a signature.

Integration of Guardtime's KSI is simplest with rsyslog, which has a separate configurable output module available. It is also possible to specify how often (after n number of log messages or after file rotation) logs are signed in Rsyslog. It is possible to request free developer access to the KSI blockchain from [83] that can be used for testing. With a test account, no additional hardware is required but it has some limitation on how many requests can be made and logs can be signed per second. When going to production, the KSI gateways should be deployed on premise. These gateways can run on a Linux operating systems such as RHEL or CentOS 7. They aggregate signature requests and

communicate with Guardtime's KSI servers/blockchain. Enterprise support is available directly from Guardtime.

Local and third party signature verification is possible. For third party verifiability, signatures need to be extended through publications. Publication frequency is once a month and it is recommended that signatures be extended as soon as new publications become available [84]. Several files and locations are needed to verify signatures. Log file (record) or hash of the file (record), signature, trust anchor type (publication, PKI signature) and trust anchor specific data [84] are needed. A signature is considered valid if signature itself is intact and the file (record) signed is the same that was used during signing. Internal verification can be done without external trust anchors.

OpenTimestamps

It is important to note that OpenTimestamps is currently in alpha and it does not cover some of the requirements set, such as enterprise support, generation of integrity proof per log line and proof of file ownership. However, it could be considered an alternative to Guardtime's KSI, once it is mature. There are differences and similarities between the two. While Guardtime's KSI creates signatures that prove both integrity and ownership, OpenTimestamps [73] does only the former. In addition, OpenTimestamps seems to be driven by two developers, possibly explaining why enterprise support is not currently available. Compared to Guardtime's KSI single permissioned blockchain that offers fast consensus, OpenTimestamps supports public Bitcoin blockchain with plans to include support for others. While OpenTimestamps can create an integrity proof per file, generating integrity proof per log line is not an option. At this point OpenTimestamps will not be considered an option due to not complying with the mandatory requirement of enterprise support and recommended requirement of generating integrity proof per log line.

Stampery

Stampery [74] is a company that provides data existence, integrity, ownership and reception proof through the use of public blockchains. They use the Bitcoin and Etherium blockchains through technology they called Blockchain Timestamping Architecture (BTA) [75]. BTA is used to anchor data to blockachains and it provides receipts (integrity proof) that can later be used to prove file existence and integrity at specific time. An API is available that can be used for integrating applications with BTA, it is also compatible with OpenTimestamps. The API requires basic authentication which can be achieved by signing up for free. The author and one of his colleague have tried to get in contact with Stampery to acquire more information about integration with syslog daemons and enterprise support. After a long wait the author was referred to the API documentation and Trailbot [76].

Trailbot is created by Stampery and enables monitoring of files with features to rollback, backup, stamp (send hash of the file to BTA), notify and more with policies. Trailbot has three components that need to be installed [76]. A client is installed on a computer that has graphical user interface and it provides means to manage watchers, policies and events. Watchers are installed on servers that have logs or files to be monitored. They enforce policies set by the client and register events. A vault can be installed as a separate server or Stampery's official vault can be used. Trailbot can use Blockchain Timestamping Smart Policy [85] to monitor a log file and embed the hash of the file into blockchain every time it is modified. A proof is returned that can later be used to prove integrity, existence and ownership of the file. While it can show which line was added when, it lacks the capability to generate integrity proof per record, has no enterprise support and is not yet mature for use in production environment.

3.3. In-depth comparison of suitable solutions

In this section comparison of the software solutions that were selected in previous sections is done. Results of the syslog analysis will be depicted in Table 4. Columns of the table will show chosen software and rows show features/plugins. Footnotes will indicate special cases like plugins that are community-maintained and due to that might lack support. If the software provides the feature, based on available documentation, then it will be indicated by a plus sign (+) and if not, then a minus sign (-) will be used.

3.3.1. Syslog solutions comparison

In Table 4, different syslog daemons are compared. Only the features that are available in stable versions, as of writing this thesis, are mentioned. Currently available (1.04.2017) software versions that will be compared are: rsyslog version 8.25, syslog-ng OSE 3.9 (syslog-ng PE 6.0.4), Fluentd version 0.12.31. Syslog-ng OSE will be used in the base comparison and features available only for PE will be indicated by identifier *PE after the plus sign.

For Fluentd there are hundreds of plugins. Most of them are created by members of the community but some are created by core committers or companies. Plugins by core committers or companies have a certificate icon on Fluentd plugins page [86]. Certified plugins should be more stable and include better support, because of that they are marked in the table by a plain plus sign. The same goes for the main Fluentd plugins mentioned

in the documentation. Other community plugins will have an asterisk (*) after the plus sign. Not all of the Fluentd plugins are listed to keep the size of the table manageable.

Information about features and plugins was gathered from available documentation and websites: [17], [24], [25], [86], [87].

Table 4. Syslog daemon comparison

Feature	**rsyslog 8.25**	**syslog-ng OSE 3.9**	**Fluentd 0.12**
PLATFORM SUPPORT			
Runs on Linux	+	+	+
Runs on Windows	-	+*PE	-
LOCAL INPUTS			
Systemd Journal	+	+	+*
Linux log socket	+	+	+
Kernel log messages	+	+	+
Windows Event Log	-	+[1]	-
File	+	+	+
Program	-	+	+
Pipe	-	+	+*
NETWORK INPUTS			
TCP	+	+	+
UDP	+	+	+
RELP	+	-	-
RLTP	-	+*PE	-
Forward (Fluentd)	-	-	+
Forward Secure (Fluentd)	-	-	+
TLS	+	+	+
IPv6	+	+	+
GSSAPI	+	-	-
RFC3195 (BEEP)	+	-	-
Scribe	-	-	+
HTTP	-	-	+
Beats[2]	-	-	+
Kafka	-	-	+
Twitter	-	-	+

Continues...

[1] Provided by syslog-ng Agent for Windows [89], has fewer features compared to syslog-ng PE.
[2] Input to receive events sent by Elastic Beats.

Table 4 – *Continues...*

Feature	**rsyslog 8.25**	**syslog-ng OSE 3.9**	**Fluentd 0.12**
FILTERING			
Message content based	+	+	+
Syslog priority based	+	+	+
Source based	+	+	+
Modify message	+	+	+
OUTPUTS			
Systemd Journal	+	-	-
Linux log socket	+	+	-
File	+	+	+
File (compression)	+	+*PE	+
File (encryption)	+	+*PE	-
Program/script	+	+	+
stdout	+	+	+
User terminal	+	+	-
Elasticsearch	+	+	+
UDP	+	+	+
TCP	+	+	+
TLS	+	+	+
RELP	+	-	-
RLTP	-	+*PE	-
Forward (Fluentd)	-	-	+
Forward Secure (Fluentd)	-	-	+
Guardtime KSI	+	-	-
Hadoop HDFS	+	+	+
Hadoop HTTP	+	-	+
HTTP	-	+	+*
HTTPS	-	-	+*
Redis	+	+	+*
Kafka	+	+	+
MSSQL	+[3]	+	+*
MySQL	+	+	+
Oracle DB	+[3]	+	-
DB2	+[3]	-	-

Continues...

[3]Provided by libdbi that has issues with some databases including this one [90].

Table 4 – *Continues...*

Feature	**rsyslog 8.25**	**syslog-ng OSE 3.9**	**Fluentd 0.12**
PostgreSQL	+	+	+*
SQLite	+	+	+*
MongoDB	+	+	+
Mail (SMTP)	+	+	+
Pipe	+	+	+*
AMQP 1.0	+	+	+*
SNMP Trap	+	-	+*
UDP spoofing	+	+	-
Graphite	-	+	+*
Loggly	-	+	+*
Logmatic.io	-	+	+*
Riemann	-	+	+*
STOMP	-	+	-
InfluxDB	-	-	+
roundrobin	-	-	+
Redmine	-	-	+
s3	-	-	+
OTHER			
Statistics about received, dropped and stored messages	+	+	+
Memory buffering	+	+	+
Disk buffering	+	+	+
Multi threading	+	+	+
Enterprise-level support	+	+	+

In the table above, the author has listed all of the input and output plugins for rsyslog and syslog-ng that were documented. Fluentd has more than 500 plugins and only those that cover what was already set by the previous syslog daemons and that have a certified status are highlighted.

For all of the solutions, especially Fluentd (because of the high number of plugins), not all of the plugins are included in the core package and some of them need to be installed separately (with dependencies) before they can be used.

For Fluentd the number of input and output plugins is higher than for the other solutions. This makes life easier when it is needed to send or receive log messages to or from specific service or software. It does not mean, however, that other syslog daemons do not support it. If TCP or UDP is used and the only requirement is a specific formatting of messages then this can be configured for both rsyslog and syslog-ng. Although plugins make reception and sending of data in various formats easier, it is not critical nor a separate requirement.

All the syslog daemons that were compared have the essential requirements set in section 3.1.1 covered. They can read log messages from files and log sockets and forward them using TCP, UDP or a custom reliable protocol. Using TLS and writing log messages to files is supported by all solutions. When encryption of written log files is needed, then every daemon (except Fluentd) provides that. All solutions also have the possibility to send log messages to local programs or scripts.

When it comes to reliable delivery of log messages, all of the solutions have their own protocols for going so. Rsyslog has RELP and syslog-ng PE has RLTP, both of which are protocols with application layer acknowledgment. Furthermore, Fluentd has its own plugins (Forward and Forward Secure) that are not compatible with others syslog daemons, similarly to RELP and RLTP. Because of this, it seems reasonable to use one syslog daemon (either rsyslog, syslog-ng or Fluentd) on both originators and collector.

Buffering of messages using memory or disk queues on the syslog daemon side is covered by all solutions. Similarly multi threading is supported by all solutions.

Enterprise support for the syslog daemons varies. Rsyslog, offered by Adiscon, has three options that all include phone and email support with different number of development hours, allowed installations and response times. For example, Unlimited Enterprise Support costing 19 999 EUR per year with unlimited number of installations, 24 hour response times on business days and 40 hours of development [91]. Syslog-ng PE has similarly three levels of support. There is no cost included but the availability of support, initial response time, software subscription and non-charged engineer hours per month vary between levels [92]. Fluentd has two options of enterprise support with no prices listed [94].

3.3.2. Integrity solutions comparison

There is only one suitable integrity solution currently available that satisfies all of the requirements. Please refer to section 3.2.2 where it is listed why Guardtime's KSI is suitable while others are not. Nevertheless, a short comparison (Table 5) will be given to provide a brief overview of some the capabilities of KSI and two other promising solutions.

Table 5. Log integrity solution comparison

	Guartime's KSI	**OpenTimestamps**	**Stampery's Trailbot**
Ready for production	yes	no	no
Integrity proof per record	yes	no	no
Integrity proof per n number of records	yes	no	no
Integrity proof per log file	yes	yes	yes
Local verification	yes	yes[4]	yes
Third party verification	yes	yes[5]	yes
Proves data existed at specific time	yes	yes	yes
Proves owner of data	yes	no	yes

KSI can be integrated to sign one or more log records or a log file by using rsyslog's output file module with signature provider options [19]. Another way is to use a KSI command line tool and specify file that will be signed. Scripting can be used to sign files after rotation or extend signatures after publication.

Trailbot can use Blockchain Timestamping Smart Policy [85] to monitor a log file and embed hash of the file into blockchain every time it is modified. A proof is returned that can later be used to prove integrity, existence and ownership of the file. While it can show which line was added when, it lacks the capability to generate integrity proof per record.

OpenTimestamps has a command line client that can be installed on a machine to create and verify timestamps with OpenTimestamps protocol [93]. This tool requires input of a

[4]Needs access to local Bitcoin Core node or public calendar server [73].
[5]Needs a local Bitcoin Core node [93].

file, so timestamping individual records is not possible.

Local verification of data is possible with KSI and Trailbot. OpenTimestamps relies on the use of either local Bitcoin Core node or connection with public calendar service, depending on how the timestamp was created.

Third party verifiable data integrity is available for all solutions. However, the time it takes to have integrity proof available that can be used by a third party is different. For OpenTimestamps and Stampery it depends on the public blockchain(s) and can take several hours. For KSI it can take up to a month, as that's how often publications are issued.

While all of the solutions prove that data existed at a specific time, only OpenTimestamps is not able to prove who the owner of the data was. While this might not be an important factor for internal use, it can become relevant in providing proof to third parties.

4. Tests

In this chapter, testing of syslog daemons and integrity solutions will be conducted. For syslog daemons, reliable log delivery is tested with default and custom configuration. Default configuration will be created to be as simple as possible based on the examples that documentation provides. Custom configurations will be created that utilize buffering and reliable transfer protocols (RELP, RLTP, Forward). During log transfer in both cases, actions (called interruptions from here on) that try to cause message loss will be introduced, such as: restarting the syslog daemon, restarting the OS, or losing network connectivity. The number of lost messages will be recorded in a table and compared to find syslog daemon with least message loss.

4.1. Lab description

The lab is built utilizing Amazon Elastic Compute Cloud (Amazon EC2) [95]. While the author has access to Swedbank's infrastructure and could build a test environment there, it has restrictions and limitations in place that make initial testing harder to start with and more time consuming. Using a cloud service makes it easier to conduct tests and enables effortless communication with third-parties (blockchain based integrity solutions).

The lab will consist of four servers: one collector and three originators. While the number of originators is rather small, it is easier to conduct testing on a limited and controlled set of servers at first. Future work could include tests in real environments, but this is outside of the scope of this thesis.

Originators are running on t2.micro instances (one CPU and one gigabyte of memory) and collector on t2.xlarge instance (4 CPUs and 16 GB of memory) [96]. CPUs are Intel Xeon E5-2676 v3 @ 2.40GHz. Operating system for all servers is Red Hat Enterprise Linux Server release 7.3 (Maipo). All the servers are in the same subnet. Port 514 will be used to deliver log messages using TCP and default configuration. Port 5514 will be used for reliability oriented protocols (RELP, RLTP). As an exception, Fluentd will use port 10514 as it was not able to listen on 514 or 5514.

Test messages will be generated into plain text files (input1.log and input2.log) on each originator. Each Originator has two inputs, one is a plain text file (input1.log) and the other is systemd journal. A script on each originator will write messages from the second

file (input2.log) into systemd journal. For rsyslog, besides testing input from systemd journal, extra tests are made with log socket as input. These tests illustrate and provide comparison between different local log sources. Log socket and systemd journal are common Linux log sources for system and application logs. Plain text files are the second most common input. That is the reason why, for each test case, two inputs have been chosen.

Figure 3 illustrates how the log messages flow from files and systemd journal through syslog daemons and end up in a file on the collector. Numbers 514, 5514 and 10514 are the port numbers used in log transfer.

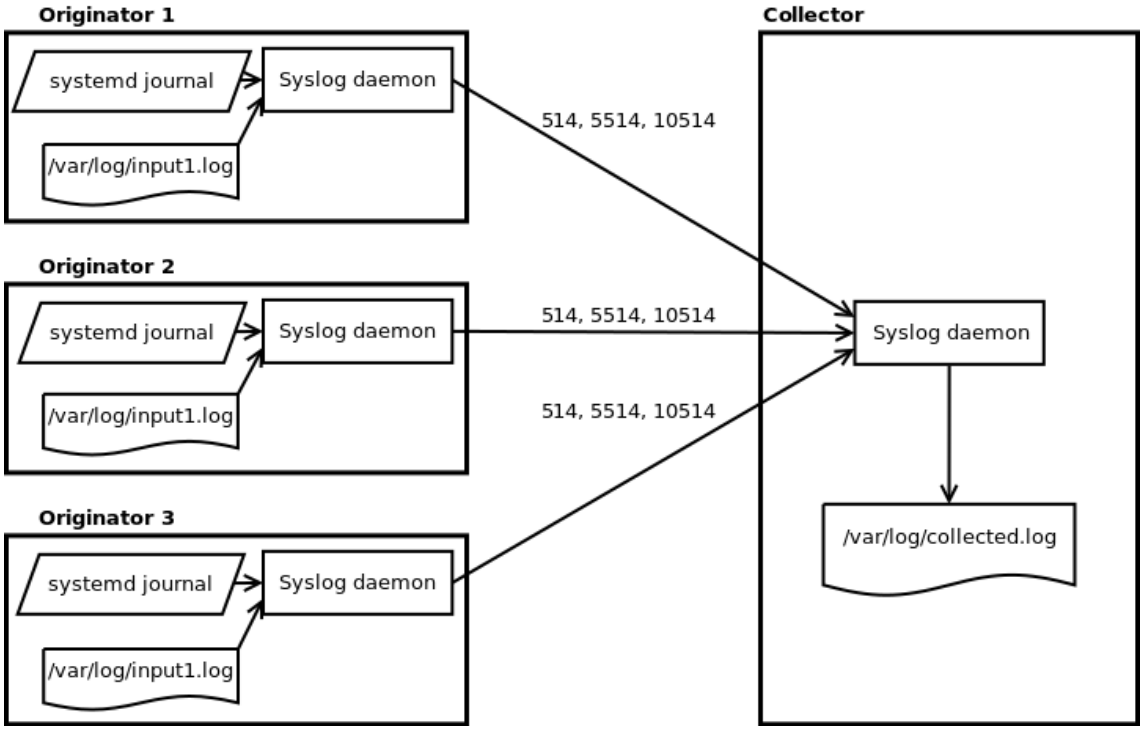

Figure 3. Message flow in lab environment

4.2. Testing log delivery

To test log delivery, a simple syslog configuration that does what is needed without specifying too many options is used. Default configuration will be created based on examples in software documentation. Different scenarios, called interrupts, will be attempted during message transfer. Examples are syslog restart, server restart, loss of network connection. A number of test cases will be run each with one interrupt. Number of lost messages will be recorded in Table 6.

After recording message loss using default configuration, a custom configuration per syslog daemon will be constructed that is focused on providing reliable delivery with reliable transport protocols (RELP, RLTP, Fluentd's forward) and message buffering enabled with modifications. The same test cases will be tried out and results recorded.

4.2.1. Test message format

During testing, a custom message format will be used to provide accurate overview of how many messages were lost and delivered. It will not be counted how many messages were duplicated. While duplication is acceptable, it is not desired, as in the long run it can increase demands on storage capacity.

The following message format is used (this example is taken from the first line of input1.log):

```
This is a test message nr 1 of 2000000 with random string
of 100 characters: MCJZlBaEpAk5ROVU4h2y96p9KTKXtqrsCM7YoH5
UQnMYAEISglJCuiJqdRf1kIVwba1GjdhWoOgBwvfm1z8x2sVwzU5tFmfz3
gRF and ID_5PGIbo
```

The message size varies from 192 to 198 characters based on the message number. Each message is numbered and one group (two million messages in this case) has unique static ID. The ID and message number are used to find out whether messages are lost and from which originators.

4.2.2. Number of test messages and verification

Some syslog daemons, depending on the configuration, send or write messages in batches. That can introduce delays when monitoring output files on the collector, as the number of messages in a batch has to be filled before they are sent or written. To avoid delays in the verification process a smaller set of messages (secondary messages) with new numbering and ID's will be generated right after the main messages.

For file input, two million test messages will be generated, followed by 5 000 secondary messages, and stored in first input file (input1.log). All of these messages are expected to be transferred to the collector, but only the main messages (two million) are counted.

Two million main messages were chosen because rsyslog took around eight seconds to transfer all of them to the collector using default configuration. That gave enough time to introduce an interrupt during log transfer.

For socket input, 100 000 messages followed by 1 000 secondary messages will be created into second file (input2.log). That file is then used by a script on the originator as input and the messages are written to systemd journal. Only the main (100 000) messages will be counted on collector. 100 000 main messages were generated based on the time it took to write them to systemd journal. The script that was used (Appendix 4.2) wrote 101 000 messages to systemd journal in around three minutes. Since the tests were not fully automatic it would have taken much longer with a higher number of messages.

Verification of the transferred and lost messages is done by a script (Apeendix 4.1) that takes six IDs (three originators each with two different IDs) as input. It returns how many messages per ID were expected and how many were lost.

4.2.3. Reliability tests

Each syslog daemon will have a default and custom configuration. Eight test runs will be carried out for each configuration and syslog daemon combination. The first run (test case 1) is to make sure no message loss occurs in normal operation. Having message loss during normal operations would void any test results. After normal operation has been verified, default configuration will be used with one interruption per test case. Default configuration uses as few parameters and options as possible without specifying any special values.

While the author has done his best to create optimal configuration providing reliable delivery, it is not a trivial task. In-depth knowledge of each solution and environment is needed. Configuration very much depends on the environment (system resources, network bandwidth/stability, number of hosts) and requirements set. It might be acceptable to lose some messages on the account of delivering others when it comes to that. Since each environment is different, configuration used in this thesis should not be copied and pasted onto production environments, but should be used as a set of recommendations that may need some additional environmental-specific fine-tuning.

Testing process

Each test case (detailed description at 4.2.4) is carried out on three originators and one collector. The steps taken before, during and after test runs are different for the originators

and the collector.

On originators, each test case was conducted according to the testing process. Testing process can be divided into several chronological stages:

1. Syslog daemon is stopped;
2. Previous log input files deleted and two new files generated. First file is input for syslog daemon, second is for a script that writes to systemd journal;
3. Syslog daemon started at specified time on all originators;
4. Script is started that reads from second input file and writes to log socket;
5. Syslog daemon starts reading messages from the file and systemd journal and forwarding them to the collector;
6. Interruption is introduced two to four seconds after syslog daemon starts;
7. After the script (started in stage 4) finishes, manual monitoring for message traffic is done for up to five minutes. When no noticeable traffic is observed then the test run is considered finished and message verification on collector can start.

Stages one to five are carried out by a script (Appendix 4.2) with the input of time (hh:mm:ss) on when to start syslog daemon. While system time is synchronized on all originators, it provides assurance that the interrupt affects all originators message transfer. Creating of test files takes roughly the same amount of time, as the originators have identical system resources. However, if one host finished a few seconds faster that others, then it can transfer some or all log messages before the interrupt occurs. For example, transferring of two million messages took eight seconds using rsylog with default configuration.

Interruption is introduced manually by observing when the syslog daemon is started, waiting a few seconds and giving the command.

On the collector, the forwarded messages are verified after there is no noticeable log transfer between originators and collector. Counting is done by a script (Appendix 4.1) that requires six IDs as input. Two IDs per host, one for log messages from file and the other for log messages from systemd journal. Using grep, cut, sort, uniq and wc commands the script returns how many lines were received and how many lost. After verification of collected log messages has been done, syslog daemon is stopped, created

log files deleted and daemon started. Cleaning of originators and collector is also done by a script (Appendix 4.3). After cleaning files from previous test case a new test case will begin according to the testing process.

4.2.4. Test cases

Test case 1
No interruption. It is expected that no messages are lost during this test. Provides a way to verify if message counting and implemented syslog configurations work.

Test case 2
Syslog daemons are restarted on the originators almost simultaneously. Parallel ssh program (pssh)[97] is used to issue commands to all originators, from collector. Command is issued two to four seconds after stage five of testing process. For example, command "sudo service rsyslog restart" was issued when rsyslog was tested.

Test case 3
Syslog daemon is restarted on the collector with a command issued from a command line interface. For example, in case of rsyslog "sudo service rsyslog restart" is issued.

Test case 4
Syslog daemon on all originators is stopped for one minute and then started. Pssh is used to transmit the command to all originators. Command used in rsyslog example is "sudo service rsyslog stop; sleep 60; sudo service rsyslog start".

Test case 5
Syslog daemon on collector is stopped for a minute and then started. Command is issued on the collector and it is the same as for test case four.

Test case 6
Network connection loss is imitated with iptables [98]. TCP traffic coming to ports 514, 5514, 10514 is dropped for one minute on the collector. After a minute the rules are removed. Following commands are issued on the collector:

```
iptables -A INPUT -p tcp --destination-port 514 -j DROP;
iptables -A INPUT -p tcp --destination-port 5514 -j DROP;
iptables -A INPUT -p tcp --destination-port 10514 -j DROP;
sleep 60;
```

```
iptables –D INPUT –p tcp ––destination–port 514 –j DROP;
iptables –D INPUT –p tcp ––destination–port 5514 –j DROP;
iptables –D INPUT –p tcp ––destination–port 10514 –j DROP;
```

Test case 7
Operating system is restarted with the reboot command on all originators. The command is distributed from collector using pssh program.

Test case 8
Operating system is restarted with the reboot command on collector.

4.2.5. Test results

Table 6 compares number of messages lost per test case, message input, syslog daemon and configuration. Lost messages from one to 9 999 are written out as is. Everything above that will have a "k" indicating a thousand. For example, 128k is 128 000 messages lost. Total number of messages received per test case from systemd journal was 300 000 and from files 6 000 000. Syslog daemons are shown in the left most column with letter D (for default) or C (for custom) behind their name, indicating which configuration was used. In the case of rsyslog, number 1 or 2 are used after letters D and C. They correspond to tests done with log socket input and systemd journal input. The upper row represents eight test cases. N/A for journal/socket in test case seven means that the results are not applicable. Script writing to systemd journal or log socket was stopped when the originator restarted, thus message loss was unavoidable.

Overview of test cases:

1. No interruption;
2. Syslog daemon on originator restarted;
3. Syslog daemon on collector restarted;
4. Syslog daemon on originator stopped for a minute;
5. Syslog daemon on collector stopped for a minute;
6. Network connection lost for a minute;
7. Originators (OS) restarted;

8. Collector (OS) restarted.

Table 6. Message loss per test case

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Rsyslog (D1)	**Socket**	0	11	15k	87k	128k	148	N/A	2387
Rsyslog (D1)	**File**	0	0	23k	0	3 744k	0	5 359k	1 957k
Rsyslog (D2)	**Journal**	0	0	5	78k	297k	74k	N/A	45k
Rsyslog (D2)	**File**	0	0	0	0	0	0	0	1 396k
Rsyslog (C1)	**Socket**	0	1 859	0	74k	0	0	N/A	0
Rsyslog (C1)	**File**	0	0	0	0	0	0	0	0

Continues...

Table 6 – *Continues...*

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Rsyslog (C2)	**Journal**	0	0	0	83k	0	0	N/A	0
Rsyslog (C2)	**File**	0	0	0	0	0	0	0	0
Syslog-ng (D)	**Journal**	0	0	65k	0	167k	137k	N/A	66k
Syslog-ng (D)	**File**	0	0	5526k	0	5298k	5519k	0	5493k
Syslog-ng (C)	**Journal**	0	0	0	7827	0	0	N/A	0
Syslog-ng (C)	**File**	0	0	0	0	0	0	0	0
Fluentd (D)	**Journal**	0	0	0	0	0	120k	N/A	0

Continues...

Table 6 – *Continues...*

		1	2	3	4	5	6	7	8
Fluentd (D)	**File**	0	0	0	0	0	0	228k	0
Fluentd (C)	**Journal**	0	0	0	0	0	0	N/A	0
Fluentd (C)	**File**	0	0	0	0	0	0	0	0

4.2.6. Rsyslog results

Installation

Installation of rsyslog is straightforward and described on rsyslog home page. Because rsyslog is the default syslog daemon on Red Hat Enterprise Linux, an upgrade was done instead of installation from scratch. The repository for version eight was added and rsyslog version 8.25 was installed with additional package rsyslog-relp-8.25.0 with the requirements it had.

Configuration

Rsyslog is capable of reading log messages from both log socket (/dev/log) and systemd journal. In rsyslog documentation log socket input is recommended to be used instead of the journal input. However, to provide comparison with syslog-ng and Fluentd, journal input will also be used. In Table 6 tests done with log socket input are indicated with D1, C1 and test done with journal input are marked as D2, C2.

Testing rsyslog did not proceed smoothly. Finding a custom configuration that worked took several tries. Using disk assisted queues on the originators resulted in re-installation of all the machines, when the queue size was mistakenly set to 2 000 000, system ran

out of memory. In addition, smaller disk assisted queues on originators produced message loss and delayed (for one minute and thirty seconds) restart or shutdown of rsyslog. Getting RELP to work took some time and and in the end, Security Enhanced Linux (SELinux)[100] was disabled to get it to work. SELinux remained disabled for all other tests and syslog daemons.

During testing rsylog had one shared base configuration on all hosts, /etc/rsyslog.conf (Appendix 1.1), and one host specific configuration, /etc/rsyslog.d/01*.conf (Appendix 1.2 and 1.3). In addition to default configuration, rsyslog statistics gathering (impstats), workdirectory and disabling escaping of control characters was defined in the base configuration. Disabling escaping of control characters was needed to correctly parse the messages written to log socket. Defining workdirectory helped rsyslog keep track of what was processed in the input file.

While systemd journal was used as input two modifications were made. Similarly to syslog-ng and Fluentd, systemd journal rate limiting was disabled as it introduced message loss when on. Second change was to increase rsyslog journal input (imjournal) rate limiting which by default turns on if 20 000 messages are read in 10 minutes. This is to protect the system from denial of service that can be introduced by corrupted journal database [99]. As more than 20 000 messages are sent in tests then turning off or increasing the rate limit has to be done. Rate limiting was increased to 500 000 messages in 60 seconds.

With the originator, custom configuration relied on the main queue with increased size of 200 000 messages. For the tests option "queue.fulldelaymark" of 5 000 was specified for the main queue. Using this option, incoming log messages from log sources are pushed back, if it is possible, when the main queue size reaches 5 000. This gives a buffer for 195 000 messages. Looking at rsyslog statistics, it was visible that during network or collector outage of one minute, the size of main queue did not reach higher than 50 000 messages. This is because reading from files could be controlled and stopped when needed and the script writing to log socket was not very fast. Due to the controlled rate of messages and duration of outage it was the best configuration to use. Log originators had limited resources available and using disk-assisted queues produced message loss when syslog daemon had to be restarted or shut down.

In the custom configuration for collector, disk-assisted memory queue was used. Defining disk-assisted queues provides benefits from fast memory queue and reliable disk queue. During normal operations memory queue is used, but upon system shutdown messages in memory queue are written to disk.

Test results

As expected, it can be seen that with default configuration a number of log messages are lost when interruptions happen. Custom configuration completely removed loss of messages gathered from files. This was expected as rsyslog, like others, is capable of stopping reading from files, recording the location where reading stopped and continuing from there after, when restart occurs.

Rsyslog was tested more than others because it was able to read log messages from both log socket and systemd journal. Example test cases were provided (indicated with D1 and C1 in Table 6) to measure message loss when log messages were read from local log socket, where it is not possible to record the location or adviced to stop message flow when syslog daemon is stopped. As expected, message loss occurred in almost all test cases. Custom configuration was able to help, but not when syslog daemon was stopped or restarted. The second batch of tests used systemd journal as the local log source. However, loss of log messages from socket (C1) remained in two cases and one case when reading from journal was used (C2). In both test cases, syslog daemon on the originator was restarted or stopped, meaning it was impossible to read log messages from log socket while rsyslog was in that state.

4.2.7. Syslog-ng results

Installation

Access to syslog-ng Premium Edition is not available by default. Demo version of syslog-ng PE was made available to the author, at very short notice, by Stallion AS [101]. During testing, syslog-ng PE version 6.0.4 (latest long term supported version that was available at the time of writing) was used.

Installation was very easy. An RPM package and a license file were made available to the author. Documentation on how to install syslog-ng PE is publicly available [102]. It was detailed and no questions or issues raised during the installation process.

Syslog-ng PE can run in a client mode without needing licenses, as it was on the originators. Running as a server, however, requires that license file be present. After restarting syslog-ng it finds the license file (if it is stored in the correct location) and writes log messages that indicate whether it is running as a client or a server. Demo license that the author received had some limitations. The number of clients was limited to 25, which was not an issue, but maximum processor cores was also limited to one, meaning performance tests might not provide relevant data and were thus not conducted.

Configuration

Syslog-ng forces the use of systemd journal on RHEL 7 systems even when reading from /dev/log is configured. In the documentation it is stated to be more reliable, compared to other source drivers, and thus it will be used [102]. This changes the outcome of tests for "socket" input, as systemd journal keeps its own database that can prevent message loss when used properly. Using systemd journal as the system log source introduced considerable loss of messages at first (not shown in Table 6), due to rate limiting on systemd journal side. Rate limiting was disabled before the tests.

Default configuration (Appendix 2.1) used on originators collects systemd journal, syslog-ng internal messages and those from input file and forwards them to the collector using TCP. The collector receives the messages and stores them in a file.

A custom configuration (Appendix 2.2) on the originator added option to try to constantly reconnect every second when connection was broken, the default is after 60 seconds. RLTP was enabled as the transport protocol and output queue size was increased from 10 000 messages to 100 000 and flow-control was enabled. For the collector, RLTP and flow-control were also enabled. Disk-based buffering can be used to provide additional reliability when needed. In this environment, disk-based buffering was not used as it was not needed and would not have offered additional reliability with the chosen test cases.

Test results

With the default configuration, syslog-ng either loses a lot of messages or does not lose any. Message loss occurs when the collector is unavailable or has shortly been unavailable (test number 3). By default, syslog-ng waits for one minute before trying to reconnect to a server. If during that period the output(destination) queue fills and sources generating messages are not stopped by flow-control, then messages are dropped. All the message loss seen for test runs with default configuration illustrate the point.

After enabling flow-control almost all message loss was removed, except for test case four, where syslog daemon on originator was stopper for a minute. It seems that the journal deleted some of the log messages before syslog-ng was able to send them. This could be prevented with configuring when journal deletes old log messages.

4.2.8. Fluentd results

Installation

Fluentd will also be referred to as td-agent, because it is the stable distribution of Fluentd provided by Treasure Data Inc, the creators of Fluentd, and it was used on test machines. Td-agent packages made installation of Fluentd effortless. Td-agent version 0.12.31 was used in testing. It was intalled using one command that downloaded all requirements and installed them. In addition, a community plugin, called systemd input plugin [103], was installed.

Configuration

When it comes to configuration it must be mentioned that Fluentd works differently from rsyslog and syslog-ng. Rsyslog and syslog-ng enable near real-time sending of messages and recording them to files line by line. Fluentd, however, works with a delay, because the way its queues work. On Fluentd, messages are collected into chunks. By default 64 chunks fit into a memory queue and the size of each chunk can be up to 8 megabytes. When the queue becomes full, the default mode is to send an exception to the input that stops the input if possible. For example, file input can be stopped by just not reading new lines until the exception has been lifted. All messages are queued before output and forwarding to output plugin occurs when either the flush_interval (60 seconds by default) is exceeded or the chunk size is reached.

An exception for TCP ports used will be made, because the author was not able to make Fluentd listen on port 514. Instead port 10514 will be used for both default and custom configuration. During testing only official plugins will be used if possible. However, since Fluentd was not able to read /dev/log socket, problem with permissions, with its official input plugin, community made systemd journal input plugin [103] is used. The first version the author tried (0.0.7) had a bug that caused the CPU to be constantly 99% utilized. Version 0.0.8 of the plugin fixed that. Additionally, Fluentd does not have an official plain TCP output plugin. Forward plugin, that works between Fluentd nodes, will be used. This provides more reliability in message delivery and thus test result with default configuration are not directly comparable to rsyslog and syslog-ng.

Fluentd's configuration is quite different from rsyslog and syslog-ng. It is harder to distinguish default configuration. Most of the default configuration was taken from documentation examples with minor additions, like changing the way names of new files are created. Some options were needed to be added to get the solution to work in the first place. For example, a file input plugin (called tail) needs to read from the start of the file,

while default is from end. During the tests td-agent is started only after writing to file has finished and no messages would be sent with the default option.

Fluentd's configuration defines sources and can specify tags per source. These tags are then used to determine what is done with the messages by defining match criteria. In the default configuration (Appendix 3.1) on originator, systemd journal and reading from file were defined as inputs and sending to collector as output. The collector received the messages and wrote them to file like this: /var/log/collected.log.20170410.log. It seems defining a output file name without date in it is not possible. In addition, by default it created a new file (collected.log.20170410_0.log, collected.log.20170410_1.log, etc) every time a batch(chunk) of messages were added. This would have made message reception verification very difficult. Append option disabled this behavior. Writing interval of messages was modified by flush_interval option.

Custom configuration added reliability features like heartbeat, collector failure detection and memory buffers on originators. On the collector memory buffer was modified and size increased.

Test results
Results show that by default Fluentd provides quite reliable delivery and message loss only happens when network traffic is dropped on the collector or originators restarted. It has to be mentioned that special message transport plugin was used during default configuration test cases, thus the results are not directly comparable to other syslog daemons tested. Also, because of the way Fluentd operates it has reliability features included by default.

Custom configuration removed all message loss. This makes Fluentd the most reliable solution.

4.3. Testing log integrity

In this section the integrity solutions that were analysed in chapter 3 are described. It will be looked at how the solutions work and examples provided on usable commands. While the author hoped to conduct more meaningful tests with the KSI solution installed at Swedbank, it was not possible at the time of writing as the installation was not functional. Thus a tryout account will be used that has limitations and due to that does not encourage running performance test. Besides Guardtime's KSI, Stampery's Trailbot and

OpenTimestamps are described, both of which are in the (pre-)alpha stage but have potential when matured.

4.3.1. Functionality tests

In this section it is looked at how each integrity solution operates and how they could be integrated with the syslog daemons discussed in this thesis. Simple tests will be conducted and examples provided. To verify an integrity proof is generated, the output will be compared to an integrity proof for a log file containing 100 lines. Size of proofs can be compared due to the manner in which proofs are constructed. The choice to use a file with 100 log lines is arbitrary, as it does not matter what the file being signed contains, since a hash is calculated and used. Using a file with millions of lines provides similar results, only change is the time it takes to calculate a hash.

Guardtime's KSI
Installation of KSI, in the case of using the trial service, was very easy. A repository was added and installation commands given. KSI Developer Guide [84] has all the information needed to get started.

The KSI command line tool can be used to sign files. When a log file with 100 lines was specified as input, it returned a signature file the size of 1 674 bytes. Integration with all syslog daemons could be done by utilizing scripting and the KSI command line tool.

Command used to sign a file, with content of ksi.conf following it:

```
ksi sign --conf /root/ksi.conf -i input.log \
-o input.log.ksig

# KSI signing service
-S http://tryout.guardtime.net:8080/gt-signingservice
--aggr-user username
--aggr-key access_key
# KSI extending service
-X http://tryout-extender.guardtime.net:8081/\
gt-extendingservice
--ext-user username
--ext-key access_key
# KSI publications file
```

```
-P http://verify.guardtime.com/ksi-publications.bin
--cnstr email=publications@guardtime.com
```

Integration of KSI is simplest and most flexible, when used for signing log messages, with rsyslog. To use KSI as the log signer, a few configuration lines need to be added to file output module in rsyslog configuration. Example is provided below.

```
action(type="omfile" file="/var/log/signed.log"
        sig.provider="ksi"
        sig.aggregator.uri="http://tryout.guardtime.net:8080
/gt-signingservice"
        sig.aggregator.user="username"
        sig.aggregator.key="access_key"
        sig.hashfunction="SHA2-256"
        sig.block.sizelimit="5")
```

In the configuration example above, rsyslog action stores all log messages in a file and generates a signature per block of five records. The signature file is stored in the same directory with the log file with added extension .ksisig (/var/log/signed.log.ksisig). Rsyslog can be configured with filters and rulesets to forward only log messages from one host, service or messages with specific content to this action. This enables creating configurations that sign important log messages more frequently (decreasing the sig.block.sizelimit value) than others.

Internal verification can be used to make sure the signature file is not corrupted and the file signed has not been changed:

```
ksi verify --ver-int -i input.log.ksig -f input.log
```

Signatures can be extended to enable third party verification through monthly publications. To extend a signature, the KSI tool is used with following command:

```
ksi extend --conf /root/ksi.conf -i input.log.ksig
 -o input.log.extended-ksig
```

A signature can be verified using publications, an example of publication code that is published every month can be seen below. Verification can be done with the KSI tool using following command [84]:

```
ksi verify --pub-str
        AAAAAA-CWYEKQ-AAIYPA-UJ4GRT-HXMFBE-OTB4AB \
        -XH3PT3-KNIKGV-PYCJXU-HL2TN4-RG6SCC-3ZGSBM \
        -i input.log.extended-ksig
```

Stampery's Trailbot

Trailbot has three components that it uses [76]. A client is installed on a computer that has graphical user interface and it provides means to manage watchers, policies and events. Watchers are installed on servers that have logs or files to be monitored. They enforce policies set by the client and register events. A vault can be installed as a separate server or Stampery's official vault can be used.

Unfortunately, as it is in pre-alpha, some of the functionality does not work as expected and the author was not able to get the stamping policy to work. Without it there is no generation of integrity proof happening and not much to talk about.

OpenTimestamps

Installation of OpentTimestamps is relatively easy with the exception of having to setup and run Bitcoin node. To install OpenTimestamps client, git, python and pip3 are needed. Specific installation instruction can be found at [73].

After installation command "./ots stamp input.log" can be used to timestamp a file. Creating a third party verifiable timestamp will take time. What is initially created is an "incomplete" timestamp, a small file (265 bytes) with extension .ots (input.log.ots) that refers to two public calendar servers for verification. These two calendar servers can later verify the file in question has not been modified. It is also possible to later upgrade the timestamp using command "./ots upgrade input.log.ots". It downloads the complete Bitcoin proof from calendars and saves it as part of the timestamp file. This removes the dependence on the calendar servers and verification can be done directly against the Bitcoin's blockchain.

OpenTimestamps provides an option to wait for Bitcoins blockchain transaction confirmation. Command "./ots –wait stamp input.log" is given to achieve this. Complete timestamp file size for the test file containing 100 records was 1070 bytes after using the command above.

To verify timestamps and file integrity, command "./ots verify input.log.ots" is given. Running a full Bitcoin node is needed to get a result.

OpenTimestamp could be used to generate integrity proofs (timestamps) that can verify if a file has been modified. Integration with syslog daemons can be done for example through scripting by calling stamp command after log files have been rotated.

4.4. Results and recommendations

In section 4.2 reliable log message delivery tests with rsyslog, syslog-ng PE and Fluentd were conducted. It can be seen why default configuration is not recommended to be used as it results in message loss in most test cases. For example, when the network connection is lost for a minute. While rsyslog and syslog-ng showed message loss in a higher number of test cases compared to Fluentd, it has to be mentioned that Fluentd's default configuration was not directly comparable. Fluentd used reliable transport protocol for both default and custom configuration test runs as it does not have an official TCP output plugin (not counting community plugins).

Reliable delivery test results with custom configuration show that all solutions can deliver log messages from files with no message loss. Both rsyslog and syslog-ng exhibited message loss from systemd journal when syslog daemon on originator was stopped for a minute. Fluentd was able to transfer log messages from systemd journal and file with no message loss and thus can be recommended as the most reliable solution. While journal input worked with syslog-ng and custom configuration showed slight message loss only when syslog daemon was stopped for a minute, rsyslog exhibited moderate message loss that could be alleviated with fine-tuning systemd journal and rsyslog configuration.

The integrity solution chosen in this thesis is Guardtime's KSI. The easiest way to integrate it with syslog daemon that also provides an option to choose when to generate integrity proof is to use rsyslog output plugin. Using rsyslog it can be defined whether a number of log records or log file is signed. In addition, KSI can be used with command line tools to sign log files after they have been rotated and thus is compatible with all syslog daemons covered in this thesis.

Recommendations on what solution to use based on example use cases are given below. It is assumed that log messages are collected from originators and integrity proof is generated on the collector.

- **Use case 1.** Log lines could be used as digital evidence and it is required to disclose as little information as possible (irrelevant log records to a legal case should not be exposed when integrity of log messages is verified by third party);
- **Use case 2.** Log lines could be used as digital evidence. Disclosing a log line to a third party with a number of immediate or preceding log records for verification is acceptable;
- **Use case 3.** Log lines could be used as digital evidence. Disclosing a log file to a third party for verification with all the log records for one day is acceptable;
- **Use case 4.** Log lines are not used as digital evidence. Internal verification is required.

For all the above use cases, log messages can be written into a plain text file on the originator and/or forwarded directly. In case they are written into a file, syslog daemons can follow the file and forward every new record to the collector. Tests show that this is the most reliable way, as no message loss was recorded when custom configuration was used. For all use cases Guardtime's KSI is used. Use cases that require third party verifiability should also extend the signatures after publication is made available each month.

Use case 1 can be achieved with using rsyslog on originators and collectors. To disclose as little information as possible, it is required to sign every log record separately (set sig.block.sizelimit="1" in example rsyslog configuration in section 4.3.1), so they can be disclosed and verified individually. This increases the demand on storage, as the number of signatures stored depends on the number of log lines signed.

Use case 2 can be similarly achieved with rsyslog on originator and collector, but the block size can be larger than one. When one log line is to be verified by a third party, then the whole block is disclosed. For example, if the block size is set to 100 and only one record is needed as evidence by third party, then all of the 100 records (one block) need to be provided with a corresponding signature file.

Use case 3 can be achieved with any syslog daemon solution covered in this thesis. Scripting can be used to sign log files after they are rotated every day.

Use case 4 can be achieved with any syslog daemon solution covered in this thesis, similarly to use case 3.

5. Summary

When organizations use centralized logging, they can be faced with problems like log message loss and log messages that cannot be trusted. Simple operations, such as restarting a syslog daemon, can result in a number of log message being lost if proper configuration is not used. In addition, while log messages are collected and stored, not much emphasis is put on providing log integrity. When users and administrators can accidentally or intentionally modify collected log messages, then none of the log messages can be trusted or used as digital evidence.

This thesis looked at and offered solutions to improve or completely remove message loss that can occur in centralized logging between log sender and log collector. While papers have been written that look at providing reliability in a low log volume environments, this thesis focuses on scenarios with higher log volumes and where offered methods (disk queues and syncing after every message) are not efficient or possible.

Providing log integrity on log collector was discussed and different options on how to achieve it were described. With the appearance of blockchain technologies a number of companies have emerged that utilize it to provide means to store data in a time ordered unforgeable chain. Unfortunately most of the solutions offered are not yet mature for production implementation. Examples are OpenTimestamps, Stampery's Trailbot and BTA.

Requirements were set that syslog daemons and integrity solution should comply with. Based on these requirements most promising applications were chosen and their functionality was compared in-depth.

A lab environment was created to test how syslog daemons compare in practice when message loss inducing interruptions were introduced during log transfer. Logs were read from both file and systemd journal. Default configuration and a custom configuration for each promising solution was made. Eight test cases were played out per syslog solution and configuration. Results were recorded in Table 6.

This thesis found that syslog daemons like rsyslog and syslog-nd Premium Edition are both excellent components for centralized logging solution aiming to provide reliable delivery and integrity of log messages. While testing with custom configuration resulted in slight message loss when reading system log messages, no loss was recorded for file input. Fluentd was the third candidate tested in the role of a syslog daemon and it produced ex-

cellent results. Test results show that no message loss occurred with custom configuration when using Fluentd.

This thesis found that Guardtime's KSI is currently the best choice for providing log integrity in a centralized logging environment. It has integration with syslog daemon (rsyslog), enterprise support, per record or log file signing, internal and third party verification with proof of ownership.

Future work would include improving syslog daemon configuration and showing how different configurations affect log messages being lost. New configurations could use fail-overs to a secondary collector that takes over when first one is unavailable. This would provide more lifelike results. In addition, it would be beneficial to add test cases with longer syslog daemon downtimes. In regards to log integrity, more thorough testing with unrestricted version of Guardtime's KSI should be conducted, especially in situations where every log record is signed.

References

[1] E. Kallqvist and J. Q. Lam, “Reliable and tamper resistant centralized logging in a high availability system – an investigation on ericsson sgsn-mme,” 2012.

[2] 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 2009 Seventh Annual Communication Networks and Services Research Conference, pp. 158–165, May 2009.

[3] B. Schneier and J. Kelsey, “Security audit logs to support computer forensics,” ACM TISSEC, vol. 2, no. 2, pp. 159–176, 1999

[4] D. Ma and G. Tsudik, “A new approach to secure logging,” in IFIP Annual Conference on Data and Applications Security and Privacy, pp. 48–63, Springer, 2008.

[5] J. Cucurull and J. Puiggalí, “Distributed immutabilization of secure logs,” in International Workshop on Security and Trust Management, pp. 122–137, Springer, 2016.

[6] A. Buldas, A. Truu, R. Laanoja, and R. Gerhards, “Efficient record-level keyless signatures for audit logs,” in Nordic Conference on Secure IT Systems, pp. 149–164, Springer, 2014

[7] K. Kent and M. Souppaya, “Guide to computer security log management,” NIST Special Publication, vol. 92, 2006.

[8] PCI Security Standards Council, "PCI DSS Quick Reference Guide," [Online]. Available: `https://www.pcisecuritystandards.org/documents/PCI%20SSC%20Quick%20Reference%20Guide.pdf`. [Accessed: 14.02.2017].

[9] R. Accorsi, “Log data as digital evidence: What secure logging protocols have to offer?,” in 2009 33rd Annual IEEE International Computer Software and Applications Conference, vol. 2, pp. 398–403, July 2009.

[10] A. S. Jose and B. A., “Automatic detection and rectification of dns reflection amplification attacks with hadoop mapreduce and chukwa,” in 2014 Fourth International Conference on Advances in Computing and Communications, pp. 195–198, Aug 2014.

[11] R. Gerhards, “The syslog protocol,” RFC5424, 2009.

[12] Rsyslog, "HOME," [Online]. Available: `http://www.rsyslog.com/`. [Accessed: 18.02.2017].

[13] Syslog-ng, "Open Source log management solution with over a million users worldwide," [Online]. Available: https://syslog-ng.org/. [Accessed: 25.02.2017].

[14] NXLog, "Log Management Solutions," [Online]. Available: https://nxlog.co/. [Accessed: 25.02.2017].

[15] Adiscon, "Adiscon Products Software for your Network" [Online]. Available: http://www.adiscon.com/en/. [Accessed: 25.02.2017].

[16] Rsyslog, "Sponsors and Sponsoring Opportunities," [Online]. Available: http://www.rsyslog.com/sponsors/. [Accessed: 25.02.2017].

[17] Rsyslog, "Welcome to Rsyslog," [Online]. Available: http://www.rsyslog.com/doc/v8-stable/. [Accessed: 25.02.2017].

[18] Rsyslog, "RELP - The Reliable Event Logging Protocol," [Online]. Available: http://www.rsyslog.com/doc/relp.html. [Accessed: 19.02.2017].

[19] Rsylog, "Keyless Signature Infrastructure Provider (ksi)," [Online]. Available: http://www.rsyslog.com/doc/master/configuration/modules/sigprov_ksi.html. [Accessed: 25.02.2017].

[20] Adiscon, "MonitorWare Agent," [Online]. Available: http://www.mwagent.com/. [Accessed: 25.02.2017].

[21] Balabit, "Homepage" [Online]. Available: https://www.balabit.com/. [Accessed: 25.02.2017].

[22] Balabit, "Syslog-ng Premium Edition," [Online]. Available: https://www.balabit.com/network-security/syslog-ng/central-syslog-server. [Accessed: 25.02.2017].

[23] Balabit, "The syslog-ng Premium Edition 5 LTS Administrator Guide Chapter 12. Reliable Log Transfer Protocol," [Online]. Available: https://www.balabit.com/documents/syslog-ng-pe-5.0-guides/en/syslog-ng-pe-guide-admin/html/concepts-rltp.html. [Accessed: 19.02.2017].

[24] Balabit, "The syslog-ng Open Source Edition 3.9 Administrator Guide," [Online]. Available: https://www.balabit.com/sites/default/files/documents/syslog-ng-ose-latest-guides/en/syslog-ng-ose-guide-admin/html/index.html. [Accessed: 25.02.2017].

[25] Balabit, "The syslog-ng Premium Edition 7 Administrator Guide," [Online]. Available: https://www.balabit.com/documents/syslog-ng-pe-7.0-guides/en/syslog-ng-pe-guide-admin/pdf/syslog-ng-pe-guide-admin.pdf. [Accessed: 25.02.2017].

[26] Elastic, "Logstash Reference," [Online]. Available: https://www.elastic.co/guide/en/logstash/5.3/index.html. [Accessed: 01.04.2017].

[27] Elastic, "The Elastic Story," [Online]. Available: https://www.elastic.co/about. [Accessed: 25.02.2017].

[28] Elastic, "Subscriptions that Go to Work for You," [Online]. Available: https://www.elastic.co/subscriptions#request-info. [Accessed: 25.02.2017].

[29] Elastic, "Lightweight Data Shippers," [Online]. Available: https://www.elastic.co/products/beats. [Accessed: 25.02.2017].

[30] J. Sissel, "The Lumberjack Protocol," [Online]. Available: https://github.com/elastic/logstash-forwarder/blob/master/PROTOCOL.md. [Accessed: 25.02.2017].

[31] Elastic, "Filebeat Lightweight Shipper for Logs," [Online]. Available: https://www.elastic.co/products/beats/filebeat. [Accessed: 25.02.2017].

[32] NXLog, "Features," [Online]. Available: https://nxlog.co/products/nxlog-enterprise-edition/features. [Accessed: 25.02.2017].

[33] Apache, "Welcome to Apache™ Hadoop®!," [Online]. Available: http://hadoop.apache.org/. [Accessed: 19.02.2017].

[34] Apache, "Welcome to Apache Chukwa!," [Online]. Available: http://chukwa.apache.org/. [Accessed: 19.02.2017].

[35] James Pearce, "Scribe," [Online]. Available: https://github.com/facebookarchive/scribe. [Accessed: 25.02.2017].

[36] A. Rabkin and R. H. Katz, “Chukwa: A system for reliable large-scale log collection.,” in LISA, vol. 10, pp. 1–15, 2010.

[37] Fluentd, "What is Fluentd?," [Online]. Available: http://www.fluentd.org/architecture. [Accessed: 25.02.2017].

[38] Treasure Data, "Fluent Bit," [Online]. Available: http://fluentbit.io/documentation/0.10/. [Accessed: 25.02.2017].

[39] Treasure Data, "Plug-and-Play Backend for Fluentd By the Inventors of Fluentd," [Online]. Available: http://get.treasuredata.com/fluentd.html. [Accessed: 25.02.2017].

[40] Apache, "Welcome to Apache Flume," [Online]. Available: https://flume.apache.org/. [Accessed: 19.02.2017].

[41] Apache, "Flume 1.7.0 User Guide," [Online]. Available: https://flume.apache.org/FlumeUserGuide.html. [Accessed: 19.02.2017].

[42] Hortonworks, "Enterprise Support Subscriptions," [Online]. Available: https://hortonworks.com/services/support/enterprise/. [Accessed: 19.02.2017].

[43] P. B. Makeshwar, A. Kalra, N. S. Rajput, and K. P. Singh, "Computational scalability with apache flume and mahout for large scale round the clock analysis of sensor network data," in 2015 National Conference on Recent Advances in Electronics Computer Engineering (RAECE), pp. 306–311, Feb 2015.

[44] P. Tangsatjatham and N. Nupairoj, "Hybrid big data architecture for high-speed log anomaly detection," in 2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 1–6, July 2016.

[45] Hewlett Packard Enterprise, "SIEM," [Online]. Available: http://www8.hp.com/uk/en/software-solutions/siem-security-information-event-management. [Accessed: 12.02.2017].

[46] IBM, "IBM Security QRadar SIEM," [Online]. Available: http://www-03.ibm.com/software/products/en/qradar-siem. [Accessed: 12.02.2017].

[47] LogRhythm, "Security Intelligence Platform," [Online]. Available: https://logrhythm.com/products/security-intelligence-platform. [Accessed: 14.02.2017].

[48] Intel security, "McAfee Enterprise Security Manager," [Online]. Available: https://www.mcafee.com/au/products/enterprise-security-manager.aspx. [Accessed: 14.02.2017].

[49] RSA, "RSA NETWITNESS® LOGS and PACKETS," [Online]. Available: https://www.rsa.com/en-us/products/threat-detection-and-response/rsa-netwitness-logs-packets. [Accessed: 14.02.2017].

[50] SolarWinds, "Log management software for security, compliance, and troubleshooting," [Online]. Available: http://www.solarwinds.com/log-event-manager. [Accessed: 14.02.2017].

[51] AlienVault, "AlienVault OSSIM: The World's Most Widely Used Open Source SIEM," [Online]. Available: https://www.alienvault.com/products/ossim. [Accessed: 14.02.2017].

[52] C. Lonvick, "The bsd syslog protocol," 2001.

[53] A. Okmianski, "Transmission of syslog messages over udp," 2009.

[54] F. Miao, Y. Ma, and J. Salowey, "Transport layer security (tls) transport mapping for syslog," tech. rep., 2009.

[55] J. Kelsey, J. Callas, and A. Clemm, "Signed syslog messages," tech. rep., 2010.

[56] D. New and M. T. Rose, "Reliable delivery for syslog," 2001.

[57] Rsyslog, "im3195: RFC3195 Input Module," [Online]. Available: http://www.rsyslog.com/doc/v8-stable/configuration/modules/im3195.html. [Accessed: 25.02.2017].

[58] Q. Tao, L. Yuan, and L. Youxun, "Design and implementation of data integrity check of subway log collection system," in 2016 IEEE International Conference on Big Data Analysis (ICBDA), pp. 1–4, March 2016.

[59] Apache, "Welcome to Apache Pig!," [Online]. Available: http://pig.apache.org/. [Accessed: 19.02.2017].

[60] D. Lang, "Building a 100k log/sec logging infrastructure.," in LISA, pp. 203–213, 2012.

[61] Cybernetica, "UXP Security Server," UXP, [Online]. Available: https://cyber.ee/en/e-government/uxp/core-components/uxp-security-server/. [Accessed: 11.02.2017].

[62] R. Accorsi, "Towards a secure logging mechanism for dynamic systems," in Proceedings of the 7th IT Security Symposium, Citeseer, 2005.

[63] R. Accorsi, "On the relationship of privacy and secure remote logging indynamic systems," in IFIP International Information Security Conference, pp. 329–339, Springer, 2006.

[64] R. Accorsi and A. Hohl, “Delegating secure logging in pervasive computing systems,” in International Conference on Security in Pervasive Computing, pp. 58–72, Springer, 2006.

[65] R. Zhang, Z. Chen, Z. Li, Y. Yang, and Z. Li, “An efficient scheme for log integrity check in security monitoring system,” in IET International Conference on Smart and Sustainable City 2013 (ICSSC 2013), pp. 198–202, Aug 2013.

[66] M. Bellare and B. Yee, “Forward integrity for secure audit logs,” tech. rep., Technical report, Computer Science and Engineering Department, University of California at San Diego, 1997.

[67] B. Schneier and J. Kelsey, “Cryptographic support for secure logs on untrusted machines.,” in USENIX Security, 1998.

[68] J. E. Holt, “Logcrypt: forward security and public verification for secure audit logs,” in Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54, pp. 203–211, Australian Computer Society, Inc., 2006.

[69] V. Stathopoulos, P. Kotzanikolaou, and E. Magkos, “A framework for secure and verifiable logging in public communication networks,” in International Workshop on Critical Information Infrastructures Security, pp. 273–284, Springer, 2006.

[70] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timon, and P. Wuille, “Enabling blockchain innovations with pegged sidechains,” 2014.

[71] Guardtime, "KSI Blockchain Technology," [Online]. Available: `https://guardtime.com/technology/ksi-technology`. [Accessed: 26.02.2017].

[72] A. Buldas, A. Kroonmaa, and R. Laanoja, “Keyless signatures infrastructure: How to build global distributed hash-trees,” in Nordic Conference on Secure IT Systems, pp. 313–320, Springer, 2013.

[73] P. Todd, "OpenTimestamps: Scalable, Trustless, Distributed Timestamping with Bitcoin," [Online]. Available: `https://petertodd.org/2016/opentimestamps-announcement`. [Accessed: 07.03.2017].

[74] Stampery, "Leaders in blockchain-based data certification," [Online]. Available: `https://stampery.com/`. [Accessed: 07.03.2017].

[75] A. S. de Pedro Crespo and L. I. C. García, “Stampery blockchain timestamping architecture (bta),” 2016.

[76] Stampery, "Trailbot," [Online]. Available: https://trailbot.io/. [Accessed: 07.03.2017].

[77] Swedbank, "About Swedbank," [Online]. Available: https://www.swedbank.ee/about/contact/start. [Accessed: 13.04.2017].

[78] Linux Foundation, "2014 Enterprise End User Trends Report," [Online]. Available: http://www.foxt.com/wp-content/uploads/2015/08/lf_end_user_report_1214-1.pdf. [Accessed: 07.03.2017].

[79] Elastic, "Persistent Queues," [Online]. Available: https://www.elastic.co/guide/en/logstash/current/persistent-queues.html . [Accessed: 07.03.2017].

[80] Elastic, "Community Beats," [Online]. Available: https://www.elastic.co/guide/en/beats/libbeat/current/community-beats.html. [Accessed: 07.03.2017].

[81] M. Heese, "Journalbeat," [Online]. Available: https://github.com/mheese/journalbeat. [Accessed: 07.03.2017].

[82] NXLog Ltd, "NXLog Enterprise Edition Reference Manual for v3.0.1862," 2016.

[83] Guardtime, "KSI Blockchain for Developers," [Online]. Available: https://guardtime.com/technology/blockchain-developers. [Accessed: 25.02.2017].

[84] Guardtime, "KSI Blockchain - Developer Guide," [Online]. Available: https://docs.guardtime.net/ksi-dev-guide/. [Accessed: 07.04.2017].

[85] A. Sánchez de Pedro, "Blockchain Timestamping Smart Policy for Trailbot," [Online]. Available: https://github.com/trailbot/stamper-policy. [Accessed: 13.04.2017].

[86] Fluentd, "List of All Plugins," [Online]. Available: http://www.fluentd.org/plugins/all#input-output. [Accessed: 25.02.2017].

[87] Fluentd, "Quickstart Guide," [Online]. Available: http://docs.fluentd.org/v0.12/articles/quickstart. [Accessed: 25.02.2017].

[88] Guardtime, "Use of a globally distributed blockchain to secure SDN," [Online]. Available: http://www.ciosummits.com/Guardtime_KSI_Use_of_a_globally_distributed_blockchain_to_secure_SDN_whitepaper_1602.pdf. [Accessed: 25.02.2017].

[89] Balabit, "WINDOWS LOG MANAGEMENT," [Online]. Available: https://www.balabit.com/network-security/syslog-ng/central-syslog-server/features/windows-eventlog. [Accessed: 07.03.2017].

[90] M. Hoenicka, "General Information," [Online]. Available: http://libdbi.sourceforge.net/ . [Accessed: 07.03.2017].

[91] Rsyslog, "Enterprise Support," [Online]. Available: http://www.rsyslog.com/professional-services/enterprise-support/. [Accessed: 25.02.2017].

[92] Balabit, "Product Support," [Online]. Available: https://www.balabit.com/network-security/syslog-ng/central-syslog-server/support. [Accessed: 07.03.2017].

[93] P. Todd, "OpenTimestamps Client," [Online]. Available: https://github.com/opentimestamps/opentimestamps-client. [Accessed: 13.04.2017].

[94] Treasure Data, "Fluentd Enterprise Edition," [Online]. Available: https://www.treasuredata.com/fluentd-enterprise/. [Accessed: 07.03.2017].

[95] Amazon, "Amazon EC2," [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 07.03.2017].

[96] Amazon, "Amazon EC2 Instance Types," [Online]. Available: https://aws.amazon.com/ec2/instance-types/. [Accessed: 07.03.2017].

[97] Pssh, "pssh(1) - Linux man page," [Online]. Available: https://linux.die.net/man/1/pssh. [Accessed: 06.04.2017].

[98] H. Eychenne, "IPTABLES," [Online]. Available: http://ipset.netfilter.org/iptables.man.html. [Accessed: 06.04.2017].

[99] Rsyslog, "imjournal: Systemd Journal Input Module," [Online]. Available: http://www.rsyslog.com/doc/v8-stable/configuration/modules/imjournal.html [Accessed: 20.04.2017].

[100] J. Morris, "SELinux Project Wiki," [Online]. Available: https://selinuxproject.org/page/Main_Page. [Accessed: 06.04.2017].

[101] AS Stallion Ltd., "Home" [Online]. Available: http://www.stallion.ee/EN/. [Accessed: 07.04.2017].

[102] Balabit, "The syslog-ng Premium Edition 6 LTS Administrator Guide," [Online]. Available: https://www.balabit.com/documents/syslog-ng-pe-6.0-guides/en/syslog-ng-pe-guide-admin/html-single/index.html. [Accessed: 06.04.2017].

[103] E. Robinson, "systemd input plugin for Fluentd," [Online]. Available: https://github.com/reevoo/fluent-plugin-systemd/tree/0.0.x. [Accessed: 06.04.2017].

Appendices

1. Appendix - Rsyslog configuration

1.1. Appendix - Rsyslog base configuration

```
#### MODULES ####
module(load="impstats"
        interval="1"
        severity="7"
        log.syslog="off"
        resetcounters="on"
        log.file="/var/log/pstats")

#imuxsock and imklog were used when reading from log socket
#( D1 and C1)
#module(load="imuxsock")
#module(load="imklog")

#PersistStateInterval and StateFile used only for C2
module(load="imjournal"
        PersistStateInterval="100"
        StateFile="/var/spool/rsyslog/journal"
        ratelimit.interval="60"
        ratelimit.burst="500000" )

#### GLOBAL ####
# Use default timestamp format
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat

# Include all config files in /etc/rsyslog.d/
$IncludeConfig /etc/rsyslog.d/*.conf

global(workdirectory="/var/spool/rsyslog")
#Escaping control characters was turned off for D1/C1
#$EscapeControlCharactersOnReceive off

#### RULES ####
# Log syslog
```

```
syslog.* /var/log/syslog
```

1.2. Appendix - Rsyslog default configuration

1.2.1. Originator

```
module(load="imfile")
input(type="imfile"
        file="/var/log/input1.log"
        tag="origin1")

action(type="omfwd"
        target="172.31.15.236"
        port="514"
        protocol="tcp")
```

1.2.2. Collector

```
module(load="imptcp")
input(type="imptcp" port="514" ruleset="Ruleset_tcp")
ruleset(name="Ruleset_tcp") {
        action(type="omfile" file="/var/log/collected.log")
}
```

1.3. Appendix - Rsyslog custom configuration

1.3.1. Originator

```
module(load="imfile")
module(load="omrelp")
main_queue(
        queue.size="200000"
        queue.fulldelaymark="5000")

input(type="imfile"
file="/var/log/input1.log"
        tag="origin1")

action(type="omrelp" name="out_relp" target="172.31.15.236"
```

```
        port="5514"
        queue.type="Direct"
        action.resumeRetryCount="-1"
)
```

1.3.2. Collector

```
module(load="imrelp")
input(type="imrelp" port="5514" ruleset="Ruleset_relp")

ruleset(name="Ruleset_RELP"
        queue.type="LinkedList"
        queue.filename="Queue_RELP"
        queue.size="7000000"
        queue.highwatermark="6900000"
        queue.fulldelaymark="6000000"
        queue.maxdiskspace="3G"
        queue.maxfilesize="64M"
        queue.saveonshutdown="on"
) {
action(type="omfile" file="/var/log/collected.log")
}
```

2. Appendix - Syslog-ng configuration

2.1. Appendix - Syslog-ng default configuration

2.1.1. Originator

```
@version: 6.0
@include "scl.conf"

# sources
source s_internal { internal(); };
source s_socket { systemd-journal(); };
source s_file { file("/var/log/input1.log" tags("origin1")); };

# destinations
destination d_syslog { file("/var/log/syslog"); };
```

```
destination d_collector { network("172.31.15.236"
transport("tcp") port(514)); };

log { source(s_internal); destination(d_syslog); };
log { source(s_internal);
        source(s_socket); source(s_file);
        destination(d_collector); };
```

2.1.2. Collector

```
@version: 6.0
@include "scl.conf"
options { keep-hostname(yes); };

source s_tcp { network(ip(172.31.15.236) port(514)); };

source s_internal { internal(); };

destination d_logs { file("/var/log/collected.log"); };
destination d_syslog { file("/var/log/syslog"); };

log { source(s_tcp); destination(d_logs); };
log { source(s_internal); destination(d_syslog); };
```

2.2. Appendix - Syslog-ng custom configuration

2.2.1. Originator

```
@version: 6.0
@include "scl.conf"
options { time-reopen(1); };

# sources
source s_internal { internal(); };
source s_socket { systemd-journal(); };
source s_file { file("/var/log/input1.log" tags("origin1")); };

# destinations
destination d_syslog { file("/var/log/syslog"); };
destination d_collector { network("172.31.15.236"
```

```
transport("rltp") port(5514) log-fifo-size(100000)); };

log { source(s_internal); destination(d_syslog); };
log { source(s_internal);
        source(s_socket); source(s_file);
        destination(d_collector); flags(flow-control); };
```

2.2.2. Collector

```
@version: 6.0
@include "scl.conf"
options { keep-hostname(yes); };

source s_tcp { network(ip(172.31.15.236) transport("rltp")
port(5514)); };
source s_internal { internal(); };

destination d_logs { file("/var/log/collected.log"); };
destination d_syslog { file("/var/log/syslog"); };

log { source(s_tcp);
        destination(d_logs);
        flags(flow-control); };
log { source(s_internal); destination(d_syslog); };
```

3. Appendix - Fluentd configuration

3.1. Appendix - Fluentd default configuration

3.1.1. Originator

```
# SOURCES
<source>
        @type systemd
        path /var/log/journal
        pos_file /var/log/td-agent/journal.pos
        tag systemd.journal
        read_from_head true
</source>
```

```
<source>
        @type tail
        path /var/log/input1.log
        pos_file /var/log/td-agent/input1.log.pos
        tag file.origin
        format none
        read_from_head true
</source>

# DESTINATIONS
#Forward to collector
<match **>
        @type forward
        <server>
                name collector
                host 172.31.15.236
                port 10514
        </server>
</match>
```

3.1.2. Collector

```
<source>
        @type forward
        port 10514
        bind 172.31.15.236
</source>

# DESTINATIONS
<match **>
        @type file
        path /var/log/collected.log
        time_slice_format %Y%m%d
        flush_interval 1s
        append true
</match>
```

3.2. Appendix - Fluentd custom configuration

3.2.1. Originator

```
# SOURCES
<source>
        @type systemd
        path /var/log/journal
        pos_file /var/log/td-agent/journal.pos
        tag systemd.journal
        read_from_head true
</source>

<source>
        @type tail
        path /var/log/input1.log
        pos_file /var/log/td-agent/input1.log.pos
        tag file.origin
        format none
        read_from_head true
        read_lines_limit 500
</source>

# DESTINATIONS
#Forward to collector
<match **>
        @type forward
        require_ack_response true
        heartbeat_type tcp
        heartbeat_interval 1s
        phi_failure_detector true
        phi_threshold 16
        hard_timeout 60s
        buffer_type memory
        buffer_queue_limit 300
        buffer_chunk_limit 2m
        flush_interval 2s
        flush_at_shutdown true
        disable_retry_limit true
        slow_flush_log_threshold 5
        buffer_queue_full_action exception
```

```
        <server>
                name collector
                host 172.31.15.236
                port 10514
        </server>
</match>
```

3.2.2. Collector

```
<source>
        @type forward
        port 10514
        bind 172.31.15.236
</source>

# DESTINATIONS
<match **>
        @type file
        path /var/log/collected.log
        time_slice_format %Y%m%d
        flush_interval 1s
        append true
        buffer_type memory
        buffer_queue_limit 900
        buffer_chunk_limit 8m
        flush_interval 2s
        flush_at_shutdown true
        disable_retry_limit true
        slow_flush_log_threshold 5
        buffer_queue_full_action exception
</match>
```

4. Appendix - scripts

4.1. Appendix - Message counting script

```
#!/usr/bin/env bash
# Name: Verify.sh
```

```
if [ $# -ne 6 ]; then
        echo $0: usage: ./Verify.sh ID_1 ID_2 ... ID_6
        exit 1
fi

MAX="2000000"
MAX_S="100000"
#FILE="/var/log/collected.log"
DATE=`date +%Y%m%d`
FILE="/var/log/collected.log.${DATE}.log"

echo "From file ${1}"
echo "From file ${2}"
echo "From file ${3}"
echo "From socket ${4}"
echo "From socket ${5}"
echo "From socket ${6}"

O1F=`cat ${FILE} | grep ${1} | grep -Eo "nr [0-9]* of" \
 | cut -d" " -f2 | sort | uniq | wc -l`
O2F=`cat ${FILE} | grep ${2} | grep -Eo "nr [0-9]* of" \
| cut -d" " -f2 | sort | uniq | wc -l`
O3F=`cat ${FILE} | grep ${3} | grep -Eo "nr [0-9]* of" \
| cut -d" " -f2 | sort | uniq | wc -l`
O1S=`cat ${FILE} | grep ${4} | grep -Eo "nr [0-9]* of" \
| cut -d" " -f2 | sort | uniq | wc -l`
O2S=`cat ${FILE} | grep ${5} | grep -Eo "nr [0-9]* of" \
| cut -d" " -f2 | sort | uniq | wc -l`
O3S=`cat ${FILE} | grep ${6} | grep -Eo "nr [0-9]* of" \
| cut -d" " -f2 | sort | uniq | wc -l`
echo "${O1F} ${O2F} ${O3F} ${O1S} ${O2S} ${O3S}"

echo -e "\nFROM FILES "
let TFILE=${MAX}*3
let FFILE=${O1F}+${O2F}+${O3F}
let LFILE=${TFILE}-${FFILE}
echo "Expected ${TFILE}, received ${FFILE}, LOST: ${LFILE} "

echo -e "\nFROM LOG SOCKETS "
let TSOCK=${MAX_S}*3
let FSOCK=${O1S}+${O2S}+${O3S}
```

```
let LSOCK=${TSOCK}-${FSOCK}
echo "Expected ${TSOCK}, received ${FSOCK}, LOST: ${LSOCK} "
```

4.2. Appendix - Message creation script

```
#!/usr/bin/env bash
# Name: Message_generator.sh

#Stop syslog
date; echo "Stopping syslog"
#sudo service rsyslog stop
#sudo service syslog-ng stop
sudo service td-agent stop

#Generate x number of main messages
#Generate y number of secondary messages
MAX="2000000"
MAX2="5000"
MAX_S="100000"
MAX_S2="1000"
#File1 is input for reading from file
#File2 is input for sending to log socket
FILE1="/var/log/input1.log"
FILE2="/var/log/input2.log"

#Remove previous files, processes and syslog spool file
rm -f ${FILE1}
rm -f ${FILE2}
#Remove syslog spool files
rm -f /var/spool/rsyslog/*
rm -f /opt/syslog-ng/var/syslog-ng.persist
rm -f /var/log/td-agent/input1.log.pos

#Check and kill other processes running this script
MYPID=$$
PIDS=($(pgrep -f Message_generator.sh|grep -v $MYPID))
for PID in $PIDS; do
        echo "Killing pid: $PID"
        kill $PID
done
echo "Removed previous processes, input and spool files"
```

```
#Creating Input1
ID=`strings /dev/urandom | grep -o "[[:alnum:]]" | head -n 6 \
 | tr -d "\n"; echo`
CHAR=`strings /dev/urandom | grep -o "[[:alnum:]]" | \
head -n 100
 | tr -d "\n"; echo`

echo "Creating new input files"
NR="1"
while [ ${NR} -le ${MAX} ]
do
echo "This is a test message nr ${NR} of ${MAX} with random \
string of 100 characters: ${CHAR} and ID_${ID}" >> ${FILE1}
NR=$[${NR}+1]
done

#Generates a smaller batch of messages
ID2=`strings /dev/urandom | grep -o "[[:alnum:]]" | head -n 6 \
| tr -d "\n"; echo`
CHAR2=`strings /dev/urandom | grep -o "[[:alnum:]]" | \
head -n 100 | tr -d "\n"; echo`

NR="1"
while [ ${NR} -le ${MAX2} ]
do
echo "This is a test message nr ${NR} of ${MAX2} with random \
string of 100 characters: ${CHAR2} and ID_${ID2}" >> ${FILE1}
NR=$[${NR}+1]
done
echo "${MAX} messages written to ${FILE1} with ID_${ID} followed \
 by ${MAX2} messages with ID_${ID2}"

#Creating Input2
ID3=`strings /dev/urandom | grep -o "[[:alnum:]]" | head -n 6 \
| tr -d "\n"; echo`
CHAR3=`strings /dev/urandom | grep -o "[[:alnum:]]" | \
head -n 100 | tr -d "\n"; echo`
TAG=origin`hostname | grep "originator" | grep -o "[0-9]"`

NR="1"
```

```
while [ ${NR} -le ${MAX_S} ]
do
echo "This is a test message nr ${NR} of ${MAX_S} with random \
string of 100 characters: ${CHAR3} and ID_${ID3}" >> ${FILE2}
#For /dev/log
#echo "<133>${TAG} This is a test message nr ${NR} of ${MAX_S} \
with random string of 100 characters: ${CHAR3} and ID_${ID3}" \
>> ${FILE2}
NR=$[${NR}+1]
done

#Generates a smaller batch of messages
NR="1"
ID4=`strings /dev/urandom | grep -o "[[:alnum:]]" | head -n 6 \
| tr -d "\n"; echo`
CHAR4=`strings /dev/urandom | grep -o "[[:alnum:]]" | \
head -n 100 | tr -d "\n"; echo`

while [ ${NR} -le ${MAX_S2} ]
do
echo "This is a test message nr ${NR} of ${MAX_S2} with random \
string of 100 characters: ${CHAR4} and ID_${ID4}" >> ${FILE2}
#For /dev/log

#echo "<133>${TAG} This is a test message nr ${NR} of ${MAX_S2} \
with random string of 100 characters: ${CHAR4} and ID_${ID4}" \
>> ${FILE2}
NR=$[${NR}+1]
done

echo "${MAX_S} messages written to ${FILE2} with ID_${ID3} \
followed by ${MAX_S2} messages with ID_${ID4}"

#Sleep and wait until all originators have created input files
if [ $# -eq 1 ]; then
TARGET=${1}
date; echo "After generating files sleeping until ${TARGET}"
current_epoch=$(date +%s)
target_epoch=$(date -d "${TARGET}" +%s)
sleep_seconds=$(( ${target_epoch} - ${current_epoch} ))
sleep ${sleep_seconds}
```

```
fi

#Start syslog
date; echo "Starting syslog"
#sudo service rsyslog start
#sudo service syslog-ng start
sudo service td-agent start

#Verify syslog started
date; echo "Verifying syslog started"
#sudo service rsyslog status
#sudo service syslog-ng status
sudo service td-agent status

#Start writing messages to log socket
echo "Starting to write messages to log socket (systemd \
journal) from ${FILE2}"

while read line
do
# socat, used to write to /dev/log
#echo "${line}" | /usr/bin/socat -t0 -T0 -lydaemon -d -\
 UNIX-SENDTO:/dev/log
# systemd-cat used to write to systemd journal
echo "${line}" | systemd-cat -t systemd_cat
done < "${FILE2}"
date; echo "Writing to log socket done"

echo "Resting for a minute"
sleep 60
date; echo "DONE"
```

4.3. Appendix - Clean script

```
#!/usr/bin/env bash
# Name: Clean.sh

#Shut down syslog
#sudo service rsyslog stop
#sudo service syslog-ng stop
sudo service td-agent stop
```

```
#Remove written log files
rm -f /var/log/input*
rm -f /var/log/collected.*

#Remove spool files
rm -f /var/spool/rsyslog/*
rm -f /opt/syslog-ng/var/syslog-ng.persist
rm -f /var/log/td-agent/input1.log.pos
rm -f /var/log/td-agent/journal.pos

#Remove journal archive and active logs,
journalctl --vacuum-size="1K" --vacuum-time="1s"
find /var/log/journal -name "*.journal" | xargs sudo rm
sudo service systemd-journal restart
chmod -R o+r /var/log/journal/

#Kill scripts that are running
MYPID=$$
PIDS=($(pgrep -f Message_generator.sh|grep -v $MYPID))
for PID in $PIDS; do
echo "Killing (Message_generator.sh) pid: $PID"
kill $PID
done

#Close previous stats collector
MYPID=$$
PIDS=($(pgrep -f "pidstat"|grep -v $MYPID))
for PID in $PIDS; do
echo "Killing (pidstat) pid: $PID"
kill $PID
done
echo "previous pidstats closed"

#Starting stats collector
#DATE=`date +%Y-%m-%d:%H:%M:%S`
#pidstat -C syslog-ng 1 >> pidstat_${DATE} &
#pidstat -C rsyslogd 1 >> pidstat_${DATE} &
#echo "New pidstats started"

#Remove configuration
```

```
#rm -f /etc/rsyslog.d/*

#Start syslog
#sudo service rsyslog start
#sudo service syslog-ng start
sudo service td-agent start

#Show last log lines
#tail /var/log/syslog
tail /var/log/td-agent/td-agent.log
```