Bachelor Thesis

Summer Semester 2013

at Fachhochschule Frankfurt am Main
University of Applied Sciences
Department of Computer Science and Engineering

towards

Bachelor of Science
Computer Science

submitted by
Jens Kühnel

Centralized and structured log file analysis with Open Source and Free Software tools

1. Supervisor: Prof. Dr. Jörg Schäfer
2. Supervisor: Prof. Dr. Matthias Schubert

topic received: 11. 07. 2013
thesis delivered: 30. 08. 2013

Abstract

This thesis gives an overview on the Open Source and Free Software tools available for a centralized and structured log file analysis. This includes the tools to convert unstructured logs into structured log and different possibilities to transport this log to a central analyzing and storage station. The different storage and analyzing tools will be introduced, as well as the different web front ends to be used by the system administrator. At the end different tool chains will be introduced, that are well tested in this field.

Revisions

Rev. 269: Official Bachelor these sent to FH

Rev. 273: Removal of Affidavit, fix of Pagenumber left/right

Table of Contents

Bibliography

ActiveMQ-Cluster: The Apache Software Foundation., ***Features > Clustering***, 2011, https://activemq.apache.org/clustering.html, retrieved: 20.08.2013

ActiveMQ-Features: The Apache Software Foundation., ***Connectivity > Cross Language Clients***, 2011, https://activemq.apache.org/cross-language-clients.html, retrieved: 20.08.2013

ActiveMQ-SSL: The Apache Software Foundation. , ***How do I use SSL***, 2011, https://activemq.apache.org/how-do-i-use-ssl.html, retrieved: 20.08.2013

AMQP: OASIS, ***AMQP A General-Purpose Middleware Standard***, 2011, http://www.amqp.org/specification/0-10/amqp-org-download, retrieved: 09.08.2013

CBE: IBM, ***Understanding Common Base Events Specification V1.0.1***, 2004, retrieved: 08.08.2013

CEEFields: MITRE Cooperation, ***CEE Core Field Dictionary***, 2012, retrieved: 13.08.2013

Churilin2013: Artyom Churilin, ***CHOOSING AN OPEN-SOURCE LOG MANAGEMENT SYSTEM FOR SMALL BUSINESS***, 2013, http://lab.cs.ttu.ee/dl135, retrieved: 20.08.2013

Chuvakin2008: Dr. Anton A. Chuvakin, ***CEE Logging Standard***, 2008, http://de.slideshare.net/anton_chuvakin/cee-logging-standard-today-and-tomorrow, retrieved: 09.08.2013

Chuvakin2013: Dr. Anton A. Chuvakin, ***Logging and Log Management***, 2013, ISBN: 978-1-59749-635-3

Czanik2013: Peter Czanik, ***PatternDB git moved and updated***, 2013, https://czanik.blogs.balabit.com/2013/05/patterndb-git-moved-and-updated/, retrieved: 08.08.2013

ELSA-UserGuide: , ***User Guide for ELSA*** , 2013, http://code.google.com/p/enterprise-log-search-and-archive/wiki/Documentation, retrieved: 24.08.2013

ELSAQuickstart: Martin Holste, ***ELSA Quickstart***, 2011, http://code.google.com/p/enterprise-log-search-and-archive/wiki/Quickstart, retrieved:

FLUENTD-FAQ: unkonwn, ***FAQ***, 2013, http://docs.fluentd.org/articles/faq, retrieved: 21.08.2013

FlumeUserGuide: The Apache Software Foundation, ***Flume 1.4.0 User Guide***, unknown, https://flume.apache.org/FlumeUserGuide.html, retrieved: 21.08.2013

FreeSoftware: Free Software Foundation, ***The Free Software Definition***, 2013, https://www.gnu.org/philosophy/free-sw.html, retrieved: 10.08.2013

GELF: Lennart Koopmann, ***Graylog Extended Log Format*** , 2011, https://github.com/Graylog2/graylog2-docs/wiki/GELF, retrieved: 09.08.2013

Gerhards2007: Rainer Gerhards, ***why does the world need another syslogd? (aka rsyslog vs. syslog-ng)***, 2007, http://blog.gerhards.net/2007/08/why-does-world-need-another-syslogd.html, retrieved: 08.08.2013

Gerhards2008: Rainer Gerhards, ***why you can't build a reliable TCP protocol without app-level acks...***, 2008, http://blog.gerhards.net/2008/05/why-you-cant-build-reliable-tcp.html, retrieved: 08.08.2013

Gerhards2011: Rainer Gerhards, ***Log Normalization Systems and CEEProfiles*** , 2011, retrieved: 07.08.2013

Gerhards2011-2: Rainer Gerhards, ***Using rsyslog mmnormalize module effectively with Adiscon LogAnalyzer***, 2011, http://www.rsyslog.com/using-rsyslog-mmnormalize-module-effectively-with-adiscon-loganalyzer/, retrieved: 08.08.2013

Gerhards2013: Rainer Gerhards, ***How to sign log messages through signature provider Guardtime***, 2013, http://www.rsyslog.com/how-to-sign-log-messages-through-signature-provider-guardtime/, retrieved: 08.08.2013

Gerhards2013-2: Rainer Gerhards, ***rsyslog's first signature provider: why Guardtime?*** , 2013, http://blog.gerhards.net/2013/05/rsyslogs-first-signature-provider-why.html, retrieved: 08.08.2013

Gheorghe2012: Radu Gheorghe, ***Using Elasticsearch for logs***, 2012, http://www.elasticsearch.org/tutorials/using-elasticsearch-for-logs/, retrieved: 08.08.2013
Gilcher2012: Florian Gilcher, ***ElasticSearch pre-flight checklist***, 2012, http://asquera.de/opensource/2012/11/25/elasticsearch-pre-flight-checklist/, retrieved: 20.08.2013
Guzdial1993: Mark Guzdial, ***Deriving Software Usage Patterns from Log Files***, 1993
HekaIntro: Mozilla Foundation, ***Introducing Heka***, 2013, https://blog.mozilla.org/services/2013/04/30/introducing-heka/, retrieved: 22.08.2013
Hintjens2013: Pieter Hintjens, ***Code Connected Volume 1 - Learning ZeroMQ***, 2013, ISBN: 1481262653
Holste2011: Martin Holste, ***Fighting APT with Open-source Software, Part 1: Logging*** , 2011, http://ossectools.blogspot.de/2011/03/fighting-apt-with-open-source-software.html, retrieved: 19.06.2013
Hwang2011: Eric Hwang, Sam Rash, ***Data Freeway: Scaling out to Realtime***, 2011, http://www.slideshare.net/slrash/2011-0630hadoopsummit-v5-8469751#btnNext, retrieved: 18.08.2013
JOURNALFIELDS: Lennart Poettering, ***systemd.journal-fields — Special journal fields***, 2012, http://www.freedesktop.org/software/systemd/man/systemd.journal-fields.html, retrieved: 13.08.2013
JOURNALJSON: Joe Rayhawk, Nis Martensen, ***Journal JSON Format***, 2013, http:/http://www.freedesktop.org/wiki/Software/systemd/json/, retrieved: 07.08.2013
Köbschall: Dr. Gerd Köbschall, ***personal interview at 22.08.2013***, 2013
Malpass2011: Ian Malpass, ***Measure Anything, Measure Everything***, 2011, http://codeascraft.com/2011/02/15/measure-anything-measure-everything/, retrieved: 09.08.2013
MSEventLog: Microsoft, ***MSDN Event Logging***, 2013, http://msdn.microsoft.com/en-us/library/aa363652.aspx, retrieved: 10.08.2013
MSEVENTSCHEMA: unknown / Microsoft, ***Windows Event Schema***, 2013, http://msdn.microsoft.com/en-us/library/windows/desktop/aa385201%28v=vs.85%29.aspx, retrieved: 14.08.2013
nxlog: Botond Botyanszki, ***NXLOG Community Edition Reference Manual for v2.5.1089*** , 2009, http://nxlog-ce.sourceforge.net/nxlog-docs/en/nxlog-reference-manual.html, retrieved: 20.08.2013
nxlog-var-warning: Botond Botyanszki, ***NXLOG Community Edition Reference Manual for v2.5.1089*** , 2009, http://nxlog-ce.sourceforge.net/nxlog-docs/en/nxlog-reference-manual.html#lang_variable_example_corr_note, retrieved: 20.08.2013
OctopussyInstallation: unknown, ***Octopussy Installation***, , http://8pussy.org/documentation/guides/administrator_guide/01_installation, retrieved: 20.08.2013
Ømq: Pieter Hintjens, ***ØMQ - The Guide*** , 2013, http://zguide.zeromq.org/page:all, retrieved: 16.08.2013
OpenSource: Open Source Iniative, ***The Open Source Definition***, unknown, http://opensource.org/osd, retrieved: 05.08.2013
OSArch2012: Amy Brown, Greg Wilson, ***The Architecture Of Open Source Applications***, 2012, ISBN: 978-1257638017
Poettering2012: Lennart Poettering, ***Forward Secure Sealing (FSS) is finally coming to +systemd's journal.***, 2012, https://plus.google.com/115547683951727699051/posts/g1E6AxVKtyc, retrieved: 08.08.2013
RabbitMQ: GoPivotal, Inc., ***What can RabbitMQ do for you?***, unknown, http://www.rabbitmq.com/features.html, retrieved: 20.08.2013
RabbitMQ-SSL: GoPivotal, Inc, ***SSL Support***, unknown, http://www.rabbitmq.com/ssl.html, retrieved: 20.08.2013
Redis: unknown, ***Introduction to Redis***, unknown, http://redis.io/topics/introduction, retrieved: 20.8.2013

redis-security: unkown, ***Redis Security***, unknown, http://redis.io/topics/security, retrieved: 20.08.2013
RELP: Rainer Gerhards, ***RELP - The Reliable Event Logging Protocol***, 2008, http://www.rsyslog.com/doc/relp.html, retrieved: 08.08.2013
RFC3164: C. Lonvick, ***RFC3164: The BSD syslog Protocol***, 2001, retrieved: 08.08.2013
RFC3339: G. Klyne, C. Newman, ***Date and Time on the Internet: Timestamps***, 2002, retrieved: 08.08.2013
RFC4627: D. Crockford, ***The application/json Media Type for JavaScript Object Notation (JSON)***, 2006, retrieved: 07.08.2013
RFC4765: H. Debar, D. Curry, B. Feinstein, ***The Intrusion Detection Message Exchange Format (IDMEF)***, 2007, retrieved: 07.08.2013
RFC5424: R. Gerhards, ***RFC5424: The Syslog Protocol***, 2009, retrieved: 08.08.2013
RFC5426: A. Okmianski, ***Transmission of Syslog Messages over UDP***, 2009, retrieved: 09.08.2013
SEC: Dr. Risto Vaarandi, ***SEC - simple event correlator***, 2013, http://simple-evcorr.sourceforge.net/, retrieved: 08.08.2013
Seguin2013: Karl Seguin, ***The Little Redis Book***, 2013, http://openmymind.net/2012/1/23/The-Little-Redis-Book/, retrieved: 14.08.2013
Shao2011: Zheng Shao, ***Real-time Analytics at Facebook***, 2011, http://www-conf.slac.stanford.edu/xldb2011/talks/xldb2011_tue_0940_FacebookRealtimeAnalytics.pdf, retrieved: 18.08.2013
Sissel2012: Jordan Sissel, ***Proposal: new logstash event schema***, 2012, https://logstash.jira.com/browse/LOGSTASH-675, retrieved: 16.08.2013
Sissel2013: Jordan Sissel, ***lumberjack***, , https://github.com/jordansissel/lumberjack, retrieved: 10.08.2013
Sissel2013-2: Jordan Sissel, ***MITRE's CEE is a failure for profit.***, 2013, http://www.semicomplete.com/blog/geekery/CEE-logging-for-profit.html, retrieved: 07.08.2013
SLES2013: SuSE Enterprise Team, ***Release Notes for SUSE Linux Enterprise Server 11 Service Pack 2***, 2013, https://www.suse.com/releasenotes/x86_64/SUSE-SLES/11-SP2/#Deprecated.future, retrieved: 17.08.2013
STOMP: Unknown, ***STOMP Protocol Specification, Version 1.2***, 2012, retrieved: 08.08.2013
Turnbull2013: James Turnbull, ***The Logstash Book***, 2013, http://www.logstashbook.com/, retrieved: 17.08.2013
ULM: J. Abela, T. Debeaupuis, ***Universal Format for Logger Messages***, 1999, https://tools.ietf.org/html/draft-abela-ulm-05, retrieved: 12.08.2013
Vaarandi2012: Dr. Risto Vaarandi,Dr. Michael R. Grimaila, ***Security Event Processingwith Simple Event Correlator***, 2012, http://ristov.users.sourceforge.net/publications/sec-issa2012.pdf, retrieved: 14.08.2013
Valdman2001: Jan Valdman, ***Log File Analysis***, 2001
XML Format: 2013, ***Extensible Markup Language (XML)*** , , retrieved:
ZMTP: iMatix, ***15/ZMTP - ZeroMQ Message Transport Protocol*** , 2012, http://rfc.zeromq.org/spec:15, retrieved: 16.08.2013
ZMTP-CURVE: Pieter Hintjens, ***26/CurveZMQ Authentication and Encryption Protocol*** , 2013, http://rfc.zeromq.org/spec:26, retrieved: 17.08.2013

Illustration Index

Index of Tables

1 Introduction

Log files are a central part in the work of a system administrator. Whenever something goes wrong, the first look is normally into one log file or another. For something so fundamental for the proper working and management of a network of computers, it is fascinating how few tools are available and how unstructured log files really are.

The only practical useful "defacto standard" was for a long time [RFC3164] which has been written to describe the syslog protocol that was and is used in different Unix systems. It offers UDP transmission to a central log file server. This syslog server only sends unstructured or semi-structured messages. In the last couple of years a strong movement came up to put log files onto a more structured path. It also has a strong emphasis on modern tools like NoSQL and JSON. This thesis will show an overview on the current state of this structured log file. It will show the different formats that are used today, which tools can help with the creation of a structured log infrastructure and what is still missing or can be improved.

1.1 Selection criteria

This thesis aims to give an overview over the available Open Source and Free Software tools with the following criteria for selecting the right tools. The criteria are based on the requirements of multiple companies the author worked in the past and present. This is aimed towards middle and larger companies.

- Combines data from many sources and different formats
  - In this thesis the enormous amount of log file analyzers that can only analyze one log file format will be ignored. This includes tools like awstat, analog etc. that only analyze apache logs.
- Easy to use for average system administrators (Windows and Linux)
  - This necessitates a usable documentation
- Has to be structured to ease analysis
- Generate structured log files out of unstructured text log files, for easy transition from unstructured to structured logs.
- Secure and reliable transport (lost messages are to be avoided, but message delivery does not need to be guaranteed)
- Data should be stored and processed redundantly to avoid single point of failure
- Fast enough to get the data from thousands of machines on a "normal" server 8-16 Cores, 16-64GB RAM
- Contains only Open Source and Free Software that can run under direct control of the user / administrator. All parts of the system must confirm to this rule.
  - Only with Open Source and Free Software it is possible to have an auditing of the used software to check for compliance with existing regulations.
  - Log files can contain personal information and should not be stored in the cloud for privacy reasons.

  - This thesis does not include OpenCore Software. OpenCore Software is software that is offered as Free and Open Source Software with special features only available in a closed source version. With OpenCore Software there are special features that make it impossible to work with, because this features are only available in a closed source version, often called "Pro" or "Enterprise". Especially encryption is not an optional feature. An OpenCore Software product will not accept a new feature like encryption, because it will hurt the sales of the closed software product. A real Open Source project will gladly accept code donations to support a new feature, like encryption.

- The tools have to analyze the data "on the fly". An analysis during the night in a batch job is to slow in a modern IT world.

- Active development is necessary, for all parts of the system. There are a lot of dead Open Source programs available, but without active development no new features and bug fixes will be available. Of course anyone could take the code and continue to develop it, but the sheer number does not allow to include them in this thesis. A project is considered dead in this thesis without a release in two years, or no commit to the code management in one year.

1.2 Programs that are included in this thesis

Program	**Language**	**stable Version**	**URL**	**License**	**Function**
logstash	Java/Ruby	1.1.13	http://logstash.net/	Apache 2.0	TCNASO
graylog2	Java/Ruby	0.12.0	http://graylog2.org	GPLv3	ASO
rsyslog	C	7.4.0	http://rsyslog.com/	GPLv3/LGPL	TCN
syslog-ng	C	3.4	http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/	LGPLv2.1/GPLv2	TCN
node-logstash	Javascript	0.0.2	https://github.com/bpaquet/node-logstash	Apache 2.0	TCNO
octopussy	perl	1.0.10	http://8pussy.org/	GPLv2/GPLv3	NSAO
nxlog	C	2.5.1089	http://nxlog-ce.sourceforge.net/	GPLv2/LGPLv2	CAN
Heka	Go	0.3.0	https://github.com/mozilla-services/heka	MPL v2.0	CN
woodchuck	Ruby	0.0.1	https://github.com/danryan/woodchuck	MIT	C
awesant	perl	0.10	https://github.com/bloonix/awesant	GPL	C
beaver	Python	30	https://github.com/josegonzalez/beaver/releases	MIT	C
lumberjack	Ruby/(C|go)	0.2.0	https://github.com/jordansissel/lumberjack/releases	Apache 2.0	C

Program	Language	stable Version	URL	License	Function
syslog-shipper	Ruby		https://github.com/jordansissel/syslog-shipper	BSD	C
remote_syslog	Ruby	1.6.4	https://github.com/papertrail/remote_syslog	BSD	C
fluentd	Ruby	1.1.15	http://fluentd.org/	Apache 2.0	C
flume	Java	1.4.0	https://flume.apache.org/	Apache 2.0	C
ncode/logix	Python	0.1.4-6	https://github.com/ncode/logix	Apache 2.0	C
systemd/journal2gelf	Python		https://github.com/systemd/journal2gelf	BSD	C
eventlog-to-syslog	C++	4.4.3	http://code.google.com/p/eventlog-to-syslog/	BSD	C
elasticsearch	Java	0.90.3	http://www.elasticsearch.org/	Apache 2.0	S
mongodb	c++	2.4.6	http://www.mongodb.org/	AGPLv3 / Apache 2.0	S
redis	C	2.6.15	http://redis.io/	BSD	T
rabbitMQ	Erlang	3.1.5	http://www.rabbitmq.com/	MPL v1.1	T
activeMQ	Java	5.8.0	https://activemq.apache.org/	Apache 2.0	T
0MQ	C++	3.2.3	http://zeromq.org/	LGPLv3+	T
SEC	perl	2.7.4	http://simple-evcorr.sourceforge.net/	GPLv2	A
Sagan	C	0.3.0	http://sagan.quadrantsec.com/	GPLv2	A
StatsD	Javascript	0.6.0	https://github.com/etsy/statsd/	MIT	O
Graphite	Python	0.9.10	http://graphite.wikidot.com/	Apache 2.0	O
Fnordmetric	Java/Ruby	0.5.1	http://fnordmetric.io/	MIT	O
Kibana2	Ruby	0.2.0	http://kibana.org/	BSD	O
Kibana3	Javascript	3.0.0-m3	http://three.kibana.org/	Apache 2.0	O
LogAnalyzer	PHP	3.6.4	http://loganalyzer.adiscon.com/	GPLv3	O

Table 1: Tools used in this thesis

About column "Functions included": T=Transport, C=Collector, N=Normalization, A=Analyzer, S=Storage, O=Output

About column "License": The author is not a lawyer and the licenses that are shown here are the ones that are shown on the website, README or LICENSE file. I did not check every file and this is not a license analysis.

1.3 What this thesis is not covering

This thesis will not cover the monitoring for availability or performance. The chances that a process dies without an error message that could be analyzed is much too high. For that kind of monitoring I suggest tools like the Open Monitoring Distribution (OMD), nagios or Zabbix.

Also not included in this thesis is the compliance with different laws and regulations, like PCI DSS, FISMA, HIPAA or best practice frameworks such as ISE2700 and COBIT. For more information see Chapter19 of [Chuvakin2013]. Also the compliance to data protection laws are not covered, even when sometimes tools for some data anonymization are shown.

Tools that are only designed to work with Intrusion Detection Systems and use only log file analysis as a small part of a large design are not described here. For this reason the tools OSSIM and ossec.net are not included.

1.3.1 Hadoop

The Hadoop ecosystem with Hadoop Distributed File System (HDFS) is the reference for Open Source big data management. The Hadoop Distributed File System is a distributed and scalable filesystem, based on the Hadoop Infrastructure and allows to use the MapReduce mechanism to let the work be done in a way to bring the computation and data storage close together, often on the same machine. To analyze, query and summarize the data the Hive Datawarehouse and the HBase non-relational Database can be used. Hadoop is an Apache project and uses the Java platform. Access to the data is available from different programming languages.

The setup and programming is quite complicated, compared to other solutions that will be covered in this thesis, but can handle much bigger datasets with multiple Petabyte. It is possible to use Hadoop to create a log analyzing infrastructure, but building a Hadoop infrastructure simply for log file analysis is oversized.

There is a hadoop subproject that creates a log analyzing platform on top of Hadoop named chuckwa, but this project is almost dead or dying with no mail in the Mailing list for 6 Month and only some one and two lines bug fixes from one developer in 2013. This does not comply with the definition of a dead project, but never the less it will not be included in this thesis, because of the necessity for hadoop.

Scribe – a tool run on top of Hadoop – was used by Facebook to analyze the log files, but this project was apparently abandoned by Facebook and replaced by a closed source tool called Calligraphus [Hwang2011] [Shao2011]

Therefore Hadoop based solutions will not be covered in the thesis, including Hive, HBase, Hadoop Distributed Filesystem, Thrift, Avro, OpenTSDB and pig.

The Apache projects flume is included because it cannot only write to Hadoop, but also to other data storages.

1.3.2 Programs that are not included in this thesis

The following programs are not included in this thesis.

Program	URL	reason for exclusion
splunk	http://www.splunk.com/	Not Open Source/Free Software

Program	URL	reason for exclusion
loggly	http://loggly.com	cloud based, not Open Source/Free Software
ntsyslog	http://ntsyslog.sourceforge.net/	dead project, last release 2007
OSSIM	http://www.alienvault.com/open-threat-exchange/projects	log managment only in Closed Source version, open core
Sguil	http://sguil.sourceforge.net/downloads.html	dead project, last release 2011
logsurfer	http://www.crypt.gen.nz/logsurfer/	dead project, last release 2011
scribe	https://github.com/facebook/scribe/	dead project, abandoned by facebook
loghound	http://ristov.users.sourceforge.net/loghound/	dead project, last release 2004
Snare	http://www.intersectalliance.com/snareagents/index.html	OpenCore, encryption only in closed source version
Bevis	https://github.com/bkjones/bevis	dead project, last commit Feb. 2012
Project Lasso	http://sourceforge.net/projects/lassolog/	dead project, last release 2008, creator under new ownership
Riemann	http://riemann.io/	only taking logs created by own log library
gelfino	https://github.com/narkisr/gelfino	dead project, last commit 2012
ossec.net	http://www.ossec.net/	log management only a very small part, no structured log files
Hadoop Tools	https://hadoop.apache.org/	See chapter 1.3.1 Hadoop
chuckwa	https://wiki.apache.org/hadoop/Chukwa	See chapter 1.3.1 Hadoop
scribe	https://github.com/facebook/scribe	See chapter 1.3.1 Hadoop
Thrift	https://thrift.apache.org/	See chapter 1.3.1 Hadoop
Avro	https://avro.apache.org/	See chapter 1.3.1 Hadoop
OpenTSDB	http://opentsdb.net/	See chapter 1.3.1 Hadoop
dendrite	https://github.com/onemorecloud/dendrite/graphs/commit-activity	No usable documentation, this project was only active for 3 week, with 8 commits all together.
loges	https://github.com/araddon/loges	No usable documentation
logtail	https://github.com/shtouff/logtail	No license attached

Program	URL	reason for exclusion
Graphtastic	https://github.com/NickPadilla/GraphTastic	No license, no commit in over a year

Table 2: Tools not used in this thesis

1.4 Structure of this thesis

This thesis is divided into 6 main chapters. The first chapter Introduction is the current chapter. It contains the selection criteria that have been used to select the programs and a short history of log files. The chapter Definitions defines the necessary basics to understand the rest of the thesis. The chapter Components and Functions shows the different parts that are needed for a centralized and structured log file analysis. The chapter Tools will show the different tools that are available in the Free Software and Open Source world and the functionalities they offer. In the chapter Toolchains the different tools will be put together to create some examples of a centralized and structured log analysis system. The chapter Conclusion will close the thesis, with a conclusion and 'a try to get a glimpse' into the future of structured log file analysis.

1.5 History of log files

Long before computers, Parish registers in which all the baptisms, marriages and burials are recorded, could be considered to be one of the first log files. A lot of similarities can be found here; One line per entry, only append and a semi-structured format.

The name log comes from the nautical log, a device that was used to measure the speed of a boat. The measurements where written into a log book, to get an overview on the progress of the journey.

In computer science the first computers used small lightbulbs to show the status of the machine and a good operator could look at the "blinkenlights" and knew the problem. When hard copy terminals were widespread the first terminal called console was used to print out the status messages of the system. When hard discs were introduced log files came into existence.

According to Dr. Gerd [Köbschall], Head of Department for Cash & Derivatives IT Operations at Deutsche Börse AG, Eschborn: "I was able to detect that the machine crashed on the changed blinking rhythm on a HP 3000. When I was working on a Control Data Corporation 1700 (created 1966) it used a teletype as a console and normally it would ring a bell when a crash occurred. Later in 1977 when OpenVMS was developed, it wrote the status messages not only to the hard copy terminal, but to the OpenVMS operator log and still does that on the current OpenVMS machines that run the Xetra systems, powering the Frankfurt Stock Exchange".

One of the first standardizations in computer log files was the creation of syslog in 1980 by Eric Allman [OSArch2012]. Initially created as a log mechanism for sendmail it was later introduced into the BSDistribution and became the de facto standard of logging in all Unix systems. But not all Unix programs are writing log messages with syslog. On a modern Linux system apache, exim and samba are three examples that are writing their own files on default, primarily for performance reasons. A lot of services are still using syslog, including most mail servers, cron, pam, inetd and ntp.

The first successful large scale introduction into structured log files was the development of the Windows Eventlog with Windows NT 3.1. The Windows Eventlog is based on a binary format, but can be queried with a .NET-based interface [MSEventLog]. In Windows Vista and Windows Server 2008 the Windows Event logging API was replaced by the Windows Event log API extending the possibilities of the API.

2 Definitions

2.1 Log file

The definition of a log file from Mark Guzdial [Guzdial1993] "discrete recordings of user actions during software use" is not universal enough, because it does not comply with most log files on Unix or Windows based systems, where also hardware and system messages are stored in log files.

Jan Valdman in [Valdman2001] uses a much wider definition: "Current software application often produce (or can be configured to produce) some auxiliary text files known as log files". This is better suited to what an average system administrator will understand under a log file, but is very vague in what is really stored in log files.

Dr. Chuvakin in [Chuvakin2013] on Page 2 defines it as: "A log messages is what a ... device ... generates in response to some sort of stimuli". The same author used a different definition at the company presentation for LogLogic in 2008 [Chuvakin2008]: "Log = message generated by an IT system to record whatever event happening".

The logstash developer Jordan Sissel defines a log message as "timestamp plus data" in [Sissel2012]. This may be a simplified view, but for the development of a log file analysis tool it is the only thing that is consistent to all log files.

My definition is a more developer referenced definition, because very often the reason for writing a log message is not understandable from the outside:

"A log file contains the information the developer of an application thought to be helpful and interesting in the current state of the software, together with the timestamp when this state occurred."

Log data or log entries or log messages are all different names for the content of a log file. Most log entries are contained in one line, but that is not valid for all log entries, like Python or Java stack traces.

2.2 Centralized log file

The definition of log file contains the word "file". This induces a reference to a normal file on the filesystem in most computer users. A program that writes to a normal log file should only append data to a log file and never change it after it is written. Security extensions like SELinux or the Windows file permissions allows to enforce the limitation to append to a log file only.

A long standing tradition in Unix networks is syslog. Syslog was written in 1980 by Eric Allman as a log mechanism for the famous sendmail program [OSArch2012]. An early add on was done to send log information to a central syslog server to have a centralized view of all information.

"Centralized" in this thesis should be defined as a way to see and query all log files of a defined group of machines (normally all machines managed by a group of people) in one webpage, database or filesystem. The log files do not need to - and should not - be stored on a single machine, instead the data should be stored on multiple machines for redundancy reasons, but the data should be the same on all machines.

2.3 Definition structured log files

Eric Allman in [OSArch2012] chapter 17.8.2 also wrote that he thinks syslog was very well designed, but specified that the only thing he should have changed was: "I would pay more attention to making the syntax of logged messages machine parseable—essentially, I failed to predict the existence of log monitoring."

The log messages are send and stored in different log formats. As Dr. Chuvakin wrote in a presentation in 2008 [Chuvakin2008]: "log format=layout of the log messages in the form of fields, separators, delimiters, tag etc."

The log file formats can be sorted into four categories as defined by R. Gerhards in [Gerhards2011]:

- "Semi-structured"
  - In this form the log entries are still largely free form, with some structures are already included, but there is "no clear distinction between field values, field delimiters and noise data".
- "weakly structured"
  - In this form the log entries are "like CSV-based formats, needs external information to understand fields".
- "strongly structured or full structured"
  - In a strongly structured log file all information stored in a structure of the format and "structured data, field names and values are provided". Possible formats are XML and JSON among others.
- "Strongly structured based on a validating profile"
  - Even a strongly structured file format can be refined when the "field names, values and semantics are provided" and can be checked. Typical problems with structured logs without validation profiles for example are:
    - undefined field names: "ip" or "ip-address" or "ipaddress"
    - format: date as defined in rfc3164 (May 23 21:22:23) or in rfc3339/ISO8601 (2013-05-23T21:22:23.24+02:00)
    - datatype: host containing only IP addresses or hostname or both

There should be a fifth category, that was not defined be R. Gerhards: "unstructured" log files. These unstructured log files are written by the Linux and the Darwin kernel as an example. Here no structure whatsoever is found.

The different formats that are in use in the field will be described later in this thesis.

2.4 Definition Open Source and Free Software

This thesis will only look at tools that are both Open Source Software as defined by the Open Source Initiative [OpenSource] and Free Software as defined by the Free Software Foundation [FreeSoftware]. The reason for that is not only price, as it is often perceived by managers and accountants, but the possibility to extend the software as the user sees fit. It also avoids the pitfalls of proprietary software, like the removing of software products by the manufacturer, therefore forcing to replace the software because of missing support/licenses or the license restriction based on log sizes.

2.5 Definition Log File Analysis

Log file analysis is the process to extract usable information from the log files. This can be statistic analysis like percentages of error messages per hosts, how many mails were sent or how often someone tried to guess a password via ssh. Also some dependency or correlation analysis could be done, like a user tried to login two times unsuccessfully and the third was successful. A third possibility is a Bayesian analysis, to detect unusual error messages.

3 Components and Functions

To create a "centralized and structured log file analysis with Free Software and Open Source tools" a lot of programs have to interact together. In this chapter the different components that are necessary will be explained.

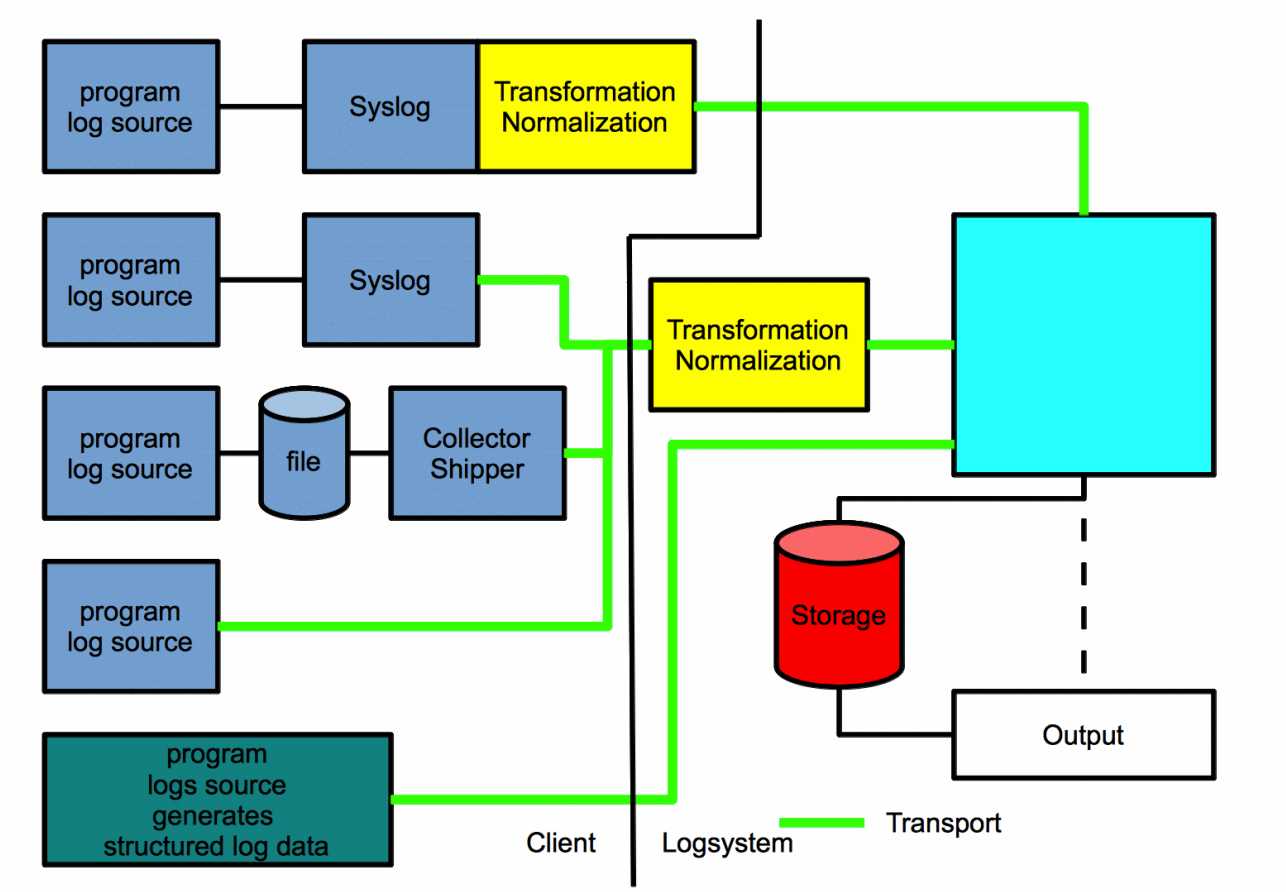

Illustration 1: Log Infrastructure

Illustration 1 shows the different ways a log message can go from the program that created the log file to the Storage and Visual Output. The first step is the creation of a log message by the program, also known as log source. The log message can be created in a lot of different formats, in chapter Formats the most common ones will be introduced.

In the chapter Transport the different mechanisms to send structured and unstructured messages are introduced. The most common used tool for storing and transporting log messages on Unix based systems is syslog. This tool is both a log format and a transport format in one.

Not all programs can send the log data using syslog. Some are only able to write into a log file on the filesystem. The Collector/Shipper will read the data from disc and forward them to the transporter or directly to the central hub.

The Transformation/Normalization step will take the log data that is not in a predefined structure and convert it into the requested structure. This can be done centralized, or on the same machine where the log source is running.

The Analysis phase tries to find problems, attacks and other abnormalities inside the log data. The Storage will save the log data and normally index it, to speed up searches. This can be done by traditional SQL systems as well as NOSQL systems. The Visual output includes graphs, but also webpages to simplify the searching and analyzing of the data.

3.1 Formats

The main problem of log files, beside the huge amount of log data that are created by all the different programs in a company, are the many different log formats. The huge amount of unstructured log files can only be handled each by itself, but the semi structured and structured log files make it possible to create rules and transformations to convert them all into a unified format. Most of the structured log files are based on one of the existing file formats JSON or XML. The [XML Format] is created by the World Wide Web Consortium, but is often considered to be too complex and hard to read for humans. The JSON format was created by Douglas Crockford and is standardized in [RFC4627] and is a very simple format that is easily read.

3.1.1 Semi structured logs

From the huge amount of semi structured logs that exist in the field only two are really defined and used. They will be shown in the next section.

3.1.1.1 BSD syslog (RFC3164)

The BSD (sometimes called traditional) syslog format was documented after being used for several years in [RFC3164]. It already showed some very simple semi structure. It contained the following fields in the header:

- PRI contains
  - Facilities (0-23)
  - Severity (0 emergency - 7 debug)
- Timestamp in the format Mmm dd hh:mm:ss (Aug 12 23:12:14)
- hostname or ip address

The RFC3164 has some very big limitations, the log entry is limited to 1024 characters and the transport is defined only for UDP and is therefore not reliable.

3.1.1.2 Modern syslog (RFC 5424)

These limitations of the traditional syslog were removed during the standardization of a new syslog format in [RFC5424] written in 2009 by R. Gerhards — developer of rsyslog — and are now used by all current syslog implementations like rsyslog and syslog-ng. The changes include removal of the 1024 byte limitation and a switch in the time format to the [RFC3339] standard, itself a sub version of ISO 8601. The RFC5424 time format looks like this:

```
2013-07-30T21:23:20.43Z
```

or

```
2013-07-30T23:23:20.43+02:00
```

The first is written in UTC, the second is written in local timezone, but both are defining the same time. Also notice the use of years and sub-second precession — another nuisance of the old syslog format. RFC5424 also added support for IPv6 addresses, for UTF-8, a required support for TLS and support for additional fields like the following:

- Version=1 (RFC5425)
- App-Name

- PROCID
- MSGID
- other structured data for example:
  - origin
    - ip address
    - enterpriseId (similar to SNMP)
    - software
    - software version
  - meta
    - sequenceID
    - sysUpTime

With RFC5424 the possibility of a structure inside syslog was added. Since the creation in 2009 a lot of implementations are already available that support RC5424. But it still is widely used to transport unstructured log data, because it does not require structured log data.

3.1.2 Structured logs

The Semi-structured log files from RFC5424 were only considered to be the first step. There are some other log file structures available. Most of the structured logs also contain the field definitions.

3.1.2.1 CEE

The Common Event Expression (CEE) is a standardization effort started in 2007 and lead by the MITRE Cooperation, a not-for-profit organization that is well known in the IT industry for the Common Vulnerabilities and Exposures (CVE) numbers.

The project created not only a standardization document that defines a log format (called CEE Profile) which is encoding neutral (called CEE Log syntax), but also a CEE Log Transport. Version 1.0-beta1 specified an event format that could be encoded in JSON or XML files. The encoding format is interchangeable, because the field names together with file types have been standardized in [CEEFields]. Three fields are required to be present: host (hostname or IP address), pname (process name) and time.

To integrate CEE into an existing syslog infrastructure a special header "@cee:" was created that is used as a prefix in front of a valid JSON message. With this prefix it is possible to send CEE messages via BSD (RFC3164) and modern syslog(RFC5425) infrastructures.

Project Lumberjack is an Open Source project to implement CEE into Open Source products. There is also a program with the same name. To differentiate the names "Project Lumberjack" and "Lumberjack" are used. Project Lumberjack is supported by both modern syslog implementations rsyslog and syslog-ng. Together with Red Hat they have spawned the projects "ceelog utils" and "libumberlog" to ease the implementation in Open Source projects. Both rsyslog and syslog-ng are ready for CEE in the current version, but beyond that there is almost no usage of CEE in the Open Source world.

```
{
 "host":"system.example.com",
 "pid":123,
 "time":"2011-12-20T12:38:05.123456-05:00",
 "msgid":"abc",
 "msg":"my event message",
 "app":"application",
 "pname":"auth",
 "sev":10,
 "action":"login",
 "status":"success"
}
```

Text 1: CEE Example log entry

All this effort looked really promising, until May 2013 when MITRE lost funding from the US government and stopped the standardization project and the mailing list. But even before this, there were already some strong opposing voices. Among others the logstash developer Jordan Sissel wrote in [Sissel2013-2] that the specifications left too many options to choose from and it is possible to create tools that are CEE compliant, but cannot talk with each other, when one is enforcing JSON and the other is enforcing XML.

3.1.2.2 GELF

[GELF] is the Graylog Extended Log Format that was created in 2010 as the native format of the graylog2 server. GELF is not a real standard, but simply the format that graylog2 specified. GELF is not only a log format, but also a transport protocol.

The definition for the GELF standard is stored in the GIT repository of graylog2 and therefore can be changed by the graylog2 developer without notice. The format also uses JSON as a file format like CEE, but also specifies that the JSON must be compressed with zlib or gzip. The following fields are specified in [GELF]:

- version: GELF spec version – "1.0" (string); MUST be set by client library.
- host: the name of the host or application that sent this message (string); MUST be set by client library.
- short_message: a short descriptive message (string); MUST be set by client library.
- full_message: a long message that can i.e. contain a backtrace and environment variables (string); optional.
- timestamp: UNIX microsecond timestamp (decimal); SHOULD be set by client library.
- level: the level equal to the standard syslog levels (decimal); optional, default is 1 (ALERT).
- facility: (string or decimal) optional, MUST be set by server to GELF if empty.
- line: the line in a file that caused the error (decimal); optional.

- file: the file (with path if you want) that caused the error (string); optional.
- _[additional field]: every other field you send and prefix with a _ (underscore) will be treated as an additional field.

```
{
  "version": "1.0",
  "host": "www1",
  "short_message": "Short message",
  "full_message": "Backtrace here\n\nmore stuff",
  "timestamp": 1291899928.412,
  "level": 1,
  "facility": "payment-backend",
  "file": "/var/www/somefile.rb",
  "line": 356,
  "_user_id": 42,
  "_something_else": "foo"
}
```

Text 2: GELF message

The GELF format is quite widely used, with support not only in graylog, but also in logstash and nxlog and others.

3.1.2.3 JSON-logstash

The logstash project with lead programmer Jordan Sissel created their own log format. This format is not really specified, but is also used by other programs as well – like fluentd and flume. In this format there are six fields that are all required, and specific extensions by the different applications are added into the "fields" field.

```
{
 "@source"    => "pork.example.com",
 "@type"      => "apache",
 "@tags"      => [],
 "@fields"    => {
  "client"       => "127.0.0.1",
  "duration_usec" => 240,
  "status"       => 404,
  "request"      => "/favicon.ico",
  "method"       => "GET",
  "referrer"     => "-"
 },
 "@timestamp" => "2012-08-22T14:53:47-0700"
}
```

Text 3: logstash JSON format

3.1.2.4 Systemd journal

The systemd is a new init system that is widely used on new Linux distributions. Systemd is the default init system for Fedora, Mandriva, OpenSuSE and many others.

One part of this system is a new log service named journal which started in 2011. The journal was again an attempt to create a new default structured log format for Linux.

The fields in the journal are separated into 3 different kinds [JOURNALFIELDS].

- Address fields (__ prefix, double underline)
  - Address fields are only usable inside the journal and should not be used outside.
- Trusted field (_ prefix, single underline)
  - Trusted fields are implicitly added by the journal and cannot be set by the log client. This includes the _PID, _UID and _EXE fields.
- User field (no prefix)
  - All other fields are user fields and can be specified by every log client itself. There are some predefined fields: MESSAGE, PRIORITY, ERRNO, CODE_FILE, CODE_LINE, CODE_FUNC, SYSLOG_FACILITY, SYSLOG_IDENTIFIER and SYSLOG_PID.

    A very important field is the MESSAGE_ID, this is a UUID field that makes it possible to generate a unique identifier for every kind of log message. All messages that are generated at the same state from the same program, should have the same message id.

The journal internally uses some binary format, but the journalctl command can export the log entries into different formats, including JSON [JOURNALJSON] and an export format. The export format can be used to send journal entries across the network and looks like a list of environment variables. The journal format allows the same field to be used multiple times inside one entry, this is mapped into a JSON array.

The systemd journal is very deeply integrated within systemd and is only available on Linux. To support other operating systems rsyslog author Rainer Gerhards created the library liblogging to create a journal replacement library that is available on all operating systems.


```
{
"_SERVICE":"systemd-logind.service"
"MESSAGE":"User harald logged in"
"MESSAGE_ID":"422bc3d271414bc8bc9570f222f24a9"
"_EXE":"/lib/systemd/systemd-logind"
"_COMM":"systemd-logind"
"_CMDLINE":"/lib/systemd/systemd-logind"
"_PID":"4711"
"_UID":"0"
"_GID":"0"
"_SYSTEMD_CGROUP":"/system/systemd-logind.service"
"_CGROUPS":"cpu:/system/systemd-logind.service"
"PRIORITY":"6"
"_BOOT_ID":"422bc3d271414bc8bc95870f222f24a9"
"_MACHINE_ID":"c686f3b205dd48e0b43ceb6eda479721"
"_HOSTNAME":"waldi"
"sLOGIN_USER":"500"
}
```


Text 4: systemd journal log entry in JSON-pretty

3.1.2.5 Windows Event Log

From NT 3.5 until Windows XP and Windows Server 2003 Windows used the Windows Event Log, internally called Event Tracing for Windows. This was replaced by a new version called Windows Eventing with Windows Vista and Windows Server 2008. Windows Eventing entries can be exported and displayed as XML and displayed with the Windows Event logs. The Event Schema XML Schema Definition is only available in the Windows SDK, but a textual description is available in [MSEVENTSCHEMA].

```
- <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
- <System>
  <Provider Name="Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-A5BA-3E3B0328C30D}" />
  <EventID>4672</EventID>
  <Version>0</Version>
  <Level>0</Level>
  <Task>12548</Task>
  <Opcode>0</Opcode>
  <Keywords>0x8020000000000000</Keywords>
  <TimeCreated SystemTime="2013-03-26T06:51:49.973366400Z" />
  <EventRecordID>2341</EventRecordID>
  <Correlation />
  <Execution ProcessID="516" ThreadID="480" />
  <Channel>Security</Channel>
  <Computer>WIN-M5PCUTLMBMT</Computer>
  <Security />
  </System>
- <EventData>
  <Data Name="SubjectUserSid">S-1-5-18</Data>
  <Data Name="SubjectUserName">SYSTEM</Data>
  <Data Name="SubjectDomainName">NT AUTHORITY</Data>
  <Data Name="SubjectLogonId">0x3e7</Data>
  <Data Name="PrivilegeList">SeAssignPrimaryTokenPrivilege SeTcbPrivilege SeSecurityPrivilege SeTakeOwnershipPrivilege SeLoadDriverPrivilege SeBackupPrivilege SeRestorePrivilege SeDebugPrivilege SeAuditPrivilege SeSystemEnvironmentPrivilege SeImpersonatePrivilege</Data>
  </EventData>
  </Event>
```

Text 5: Windows Eventlog XML file

3.1.2.6 Auditlog

The audit log is the log file of the Linux auditing system. With the help of the auditing system a linux system administrator can monitor file changes, logins, logouts, successful and unsuccessful authentications, SELinux violations and can even trace every available syscall and the result of this syscall.

Normally the auditing system writes its logs to /var/log/audit/audit.log and uses this format:

type=USER_AUTH msg=audit(1375462342.487:133952): user pid=17126 uid=0 auid=4294967295 ses=4294967295 subj=system_u:system_r:local_login_t:s0-s0:c0.c1023 msg='op=PAM:authentication acct="root" exe="/bin/login" hostname=? addr=? terminal=tty2 res=success'

Text 6: Auditlog

3.1.2.7 Intrusion Detection Message Exchange Format (IDMEF)

Intrusion Detection Message Exchange Format (IDMEF) is an XML based file format that is defined in [RFC4765], but is still in the experimental phase. This format can be used by snort and suricata.

3.1.3 Other formats

There are a lot of other formats defined, almost every company has created its own log format. Here is a list of other log formats that are known, but deemed too unimportant to describe them here in detail.

- Common Event format
  - A definition created by the company arcsight that uses a pipe (|) to separate fields.
- Security Device Event Exchange (SDEE)
  - created by Cisco as an XML based log format for their intrusion prevention system.
- Common Base Event ([CBE])
  - is a log format created by IBM and based also on XML.
- Universal Format for Logger Messages ([ULM])
  - This IETF format uses a space separated list of key=value fields and was created in 1999 as an internet draft, but was deprecated in the same year.
- other log formats
  - Apache
    - format of the access log can be specified very freely, including writing JSON-based format into the access log.
    - The format of the error log cannot be changed.
  - log4j (tomcat, JBoss)
    - log4j is an Open Source Java library that is used by a lot of Java programs, incl. tomcat and JBoss. With the help of the plugin architecture it is very easy to add support for different structured log files.
  - mod_security
    - mod_security is a web application firewall based on apache, nginx or IIS. It uses its own log format with multiple footer, headers, trailers and body. There are special tools to manage this, like AuditConsole from jwall.org.

- Python
  - Python comes with its own build-in logging class. Because Python also ships with JSON since 2.6, it is quite easy to write JSON log files with any Python program.
- Ruby
  - Ruby also comes with its own build-in logging class and changing the log format to JSON is possible with the extension (also known as gems) named logging by TwP.

3.2 Collector/Shipper

A lot of programs support syslog which directly allows it to send log data to a central log server, but not all. To support these kind of programs a collector or shipper is needed.

3.2.1 File

Most shipper or collector tools are reading existing log files line by line and send those to a central log server via a predefined transport. To speed up the detection of new lines in the log file, most tools are using a mechanism like inode notification or sized based change detection to avoid rereading every log file entry again.

3.2.2 Sockets, named pipes and STDIN

Another possibility to get log entries into the shipper are sockets and named pipes. The advantage is that no disc I/O is necessary to get the log entries into the shipper, but when the shipper is not running the log messages cannot be handled and messages could be lost. Another possibility is reading the messages from STDIN. With this mechanism the log source starts the shipper as a subprocess and sends the messages via STDIN (file descriptor 0). The log source should check if the shipper is still running and restart the shipper when necessary.

Whatever mechanism is used the log shipper sends the collected log messages via one or more of several transport mechanisms to the next step of the log analyzing toolset.

3.2.3 Local Windows Eventlog

Some shippers, when run on a Windows system, can read the Windows Event log and send them to a central server. Because the Event log is already structured it should be avoided to send the entries in an unstructured log format. But sadly enough that is what most tools do.

3.2.4 Compare collector / shipper

Programm	flat file	directory structure multiple files with *	STDIN/STDOUT	unix domain socket	named pipe	eventlog local Windows	systemd journal	spool message during downtime
logstash	rw	rw	rw	rw	rw	r		
rsyslog	rw	rw	rw	rw	rw		rw	rw
syslog-ng	rw		rw	rw	rw			

Programm	flat file	directory structure multiple files with *	STDIN/STDOUT	unix domain socket	named pipe	eventlog local Windows	systemd journal	spool message during downtime
node-logstash	rw		rw					
nxlog	rw		rw	rw		r		
Heka	rw	r	rw					
woodchuck	r	r	w					
awesant	r	r	w	rw				
beaver	r	r	w	w				
lumberjack	r	r	r					
syslog-shipper	r	r						
remote_syslog	r	r						
fluentd	rw		rw					
flume	rw	r	r	r				
systemd/journal2gelf							r	
eventlog-to-syslog						r		

Table 3: collector/shipper Overview

3.3 Transport

There are different transport mechanisms to bring the log messages to a central server. The transport defines the wire protocol that is used to send the messages. Because of the sensitive nature of log files they should be encrypted when send between machines. Not all transport mechanism are supporting that. It should also support reliable transport making sure that no message is lost on the way.

3.3.1 Syslog

The BSD syslog standard [RFC3164] also includes a transport protocol based on UDP and uses port 514. The replacement [RFC5424] does not include a transport protocol itself, but requires all implementation to support Transport Layer Security (TLS) with TCP-Port 6514. A UDP based transport using port 514 is described in [RFC5426]. These syslog protocols are the most used log transports in the field.

With the help of TCP sessions the transport of the log messages is quite reliable, but there are cases in which the loss of data sent via tcp cannot be avoided. The rsyslog author Rainer Gerhards describes this in his blogpost [Gerhards2008]. This problem led to the development of [RELP], that implements app level acknowledgment. This creates a much more reliable transport, but is missing encryption. RELP with encryption is on the TODO list of Gerhards, but is not yet stable. Until then it would be possible to use stunnel to add TLS encryption on top. The problem with RELP is that it is no standard and is not used in any other tool, like syslog-ng.

3.3.2 AMQP

The Advanced Message Queuing Protocol [AMQP] is an open standardized message middleware application layer. AMQP was developed for the financial industry, but is now used for a lot of different purposes. Its main advantages are interoperability. AMQP is an application layer protocol and it is possible to let multiple AMQP servers (aka. AMQP Broker) from different vendors talk with each other, similar to http or smtp. The other main advantage is its reliability, because it can be very tightly controlled that no message that was entered into an AMQP system can be lost.

AMQP uses special terms to describe its components: An Exchange is where the log messages are send from, where they are "produced". The Queue is where the log messages are read from. The Bindings are connecting Exchanges and Queue with one another. The Broker is the AMQP Server.

AMQP supports both username and password authentication as well as SASL authorization. It also supports TLS encryption, see Part 5 of the [AMQP] standard.

AMQP is used by the following message server: Apache Qpid, Apache ActiveMQ, RabbitMQ as well as others.

3.3.3 STOMP

[STOMP] or Simple (or Streaming) Text Orientated Messaging Protocol is a protocol similar to AMQP, but instead of being a binary format, it uses a text based format very similar to http. It is so simple that a telnet session is enough for some basic usages. Because of the text based format it is very verbose and takes much more bandwidth than necessary. It also lacks some features that are available in AMQP.

STOMP in the current version 1.2 supports username and password authentication, but encryption is not available. It is possible to use an stunnel to put an encryption layer around STOMP.

Stomp is used by two servers that are also speaking AMQP: Apache ActiveMQ and RabbitMQ(with Plugin)

3.3.4 Ømq/ZMTP

[Ømq] also known as ZeroMQ or 0MQ is another messages queue system. Unlike STOMP and AMQP it is not built for interoperability, but there are multiple implementations available. Ømq is a library that does not need a dedicated broker and is designed to be very easy to use and very fast. This simplifies setup enormously and is the main reason why it is used quite often for log transport.

The transport protocol is called [ZMTP], but is not widely used outside the Ømq project itself.

A new version of Ømq called CurveMQ was created in 2013 to bring encryption support to Ømq, the new transport protocol is named [ZMTP-CURVE], but it is very new and no stable release has been created yet.

3.3.5 Redis

Redis is a key-value store and belongs to the so called NoSQL databases. For the use as a log transport the build-in feature called "channels" is used to create a publish-subscribe messaging infrastructure. The redis transport does not support encryption. Only a password authentication scheme without usernames is available. More in the redis chapter on page 43.

3.3.6 Lumberjack

Lumberjack is the transport protocol used by the shipping tool with the same name. This is not to be confused with the project lumberjack, belonging to the CEE initiative. The developer is Jordan Sissel who also created logstash and was created because he needed a transport protocol that supported "encrypted, trusted, compressed, latency-resilient, and reliable transport of events"[Sissel2013]. More about lumberjack on page 45.

3.3.7 Remote Windows Eventlog

Microsoft Windows also has its own transport mechanism. This mechanism is primary used by Microsoft itself, but access to the Client component is available via an API on a Windows Server. It is therefore possible to collect all Eventlogs from all machines in a complete Windows domain and send it to the central machine without having to install the shipper on all machines. Sadly no Open Source and Free Software tool supports this at the moment. It was part of Project Lasso, but this project is dead now and does not support the new Log Format used since Windows Vista and Server 2008.

3.3.8 Compare Transports

Programm	BSD syslog udp (RFC3164)	IETF syslog udp (RFC5424)	IETF syslog tcp (RFC5424)	IETF syslog tcp tls (RFC5425)	TLS encrypted channel	RELP	http	websocket	redis	amqp (QPID,ActiveMQ,RabbitMQ)	Stomp (ActiveMQ,RabbitMQ)	0MQ	gelf	lumberjack	SNMP	statsD	graphite	graphtastic	varnishlog	kafka
logstash	rw	rw	rw		r	r	w	rw	rw	rw	rw	rw	rw	rw	r	w	rw	w	r	
graylog2	r	r	r	r			r			rw			r							
rsyslog	rw	rw	rw	rw	rw	rw			w			w			w					
syslog-ng	rw	rw	rw	rw	rw					w										
node-logstash	r						w		rw			rw	w			w				
octopussy	r	r	r	r	r	r														
nxlog	rw	rw	rw	rw	rw								w							
Heka	r	r	r				r			rw							w			
woodchuck									w											
awesant									w	w	w									
beaver									w	w	w	w								
lumberjack														w						
syslog-shipper	w				w															

Programm	BSD syslog udp (RFC3164)	IETF syslog udp (RFC5424)	IETF syslog tcp (RFC5424)	IETF syslog tcp tls (RFC5425)	TLS encrypted channel	RELP	http	websocket	redis	amqp (QPID,ActiveMQ,RabbitMQ)	Stomp (ActiveMQ,RabbitMQ)	0MQ	gelf	lumberjack	SNMP	statsD	graphite	graphtastic	varnishlog	kafka
remote_syslog	w	w			w															
fluentd	r						rw	w	w	rw		w			r					w
flume	r	r	r				r			r										
ncode/logix	r									w										
systemd/journal2gelf													w							
eventlog-to-syslog	w																			

Table 4: Transport Overview

3.4 Transformation/Normalization

Most of the log messages that are sent today are not yet in a structured log format. To bring structure into these log messages a transformation or normalization is necessary. This detects the different kind of messages and creates key-value pairs out of the unstructured log messages.

There are different ways to do this. A regular expression (regex) based system could be used to achieve this, but managing and writing large regular expressions can be cumbersome and error prone. Most of the tools are working on a different basis. This approach is called samples based or pattern based. Here the parsing is done based on fixed strings and matching is done with predefined field type.

The regular expression to parse a line like this:

```
sshd[1738]: Accepted password for root from 172.16.242.1 port
50447 ssh2
```

Would be:

```
sshd\[[0-9]+\]: Accepted (gssapi(-with-mic|-keyex)?|rsa|dsa|
password|publickey|keyboard-interactive/pam) for [^[:space:]]+
from [^[:space:]]+ port [0-9]+ ssh2
```

It is easier to maintain a ruleset like this:

```
sshd [!PID!]: Accepted !AUTHMETHOD! for !USERNAME! from !IP-
ADDRESS! port !PORTNUMBER! ssh2
```

The Transformation can be done on the central server or on every client itself. The central Transformation has the advantage that the rules are stored in one place and can be changed quite easily, but the CPU usage can be a problem in larger setups. To avoid that, the Normalization can be

spread out to multiple nodes, or the work can be done on the client side before the transport. The CPU load on every client is quite small, but the distribution of the rule set can be problematic, if no configuration management like puppet, chef or ansible is already in place.

3.4.1 Pattern-DB

The Pattern-DB is part of syslog-ng and is normally compiled into the syslog-ng binary. The documentation of Pattern-DB is very complete and syslog-ng has a GIT repository where it collects rules for different services. The patterns itself are stored inside an XML structure and include test messages and examples. The Pattern-DB is very actively maintained and also allows for messages to be correlated. This allows for mail server to save sender and recipient of a mail into one log entry or to put together the correlations between the logon and logoff times, to save the duration of a login.

To parse the example from above the rule should look like this:

```
sshd [@NUMBER:PID:@]: Accepted @QSTRING:auth_method:@ for
@QSTRING:username:@ from\ @QSTRING:client_addr:@ port
@NUMBER:port:@ ssh2
```

3.4.2 Liblognorm

The Liblognorm tool is developed by the rsyslog creator Rainer Gerhards and is created as a library so other tools can use this normalization tool as well. Liblognorm includes a small tool called "normalize" to check the rulesets and creates JSON messages out of normal log files. This makes rule writing much easier. The documentation is somewhat limited, but enough to create the rules, but there is no adequate rule library so all rules have to be created by oneself.

Liblognorm is not only used by rsyslog but also by the Sagan project. They have created a rule library, the only one available.

```
rules=sshd [%pid:number%:] Accepted %auth_method:word% for
%username:word% from %src-ip:ipv4% port %src-port:number% ssh2
```

3.4.3 Octopussy

Octopussy is a log management system that uses its own log normalization. The rule base is quite extensive, but it can only be used by the Octopussy system, because it is an integrated part. The patterns are stored in an XML file and can be edited and created with the help of the Octopussy webpage. This makes it very easy for a system administrator to create new patterns.

The example line from above would be found by this rule:

```
<@REGEXP("ssh\S+"):daemon@>[<@PID:pid@>]: <@REGEXP("Accepted
password for .+"):msg@>
```

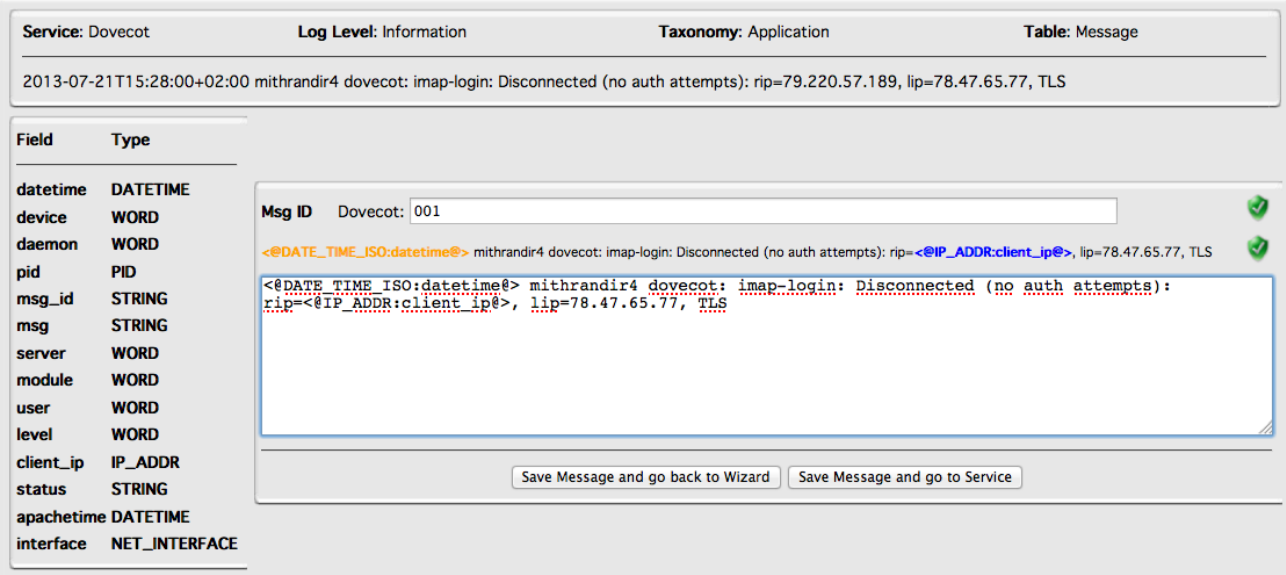

Illustration 2: Octopussy rule creation

3.4.4 Grok

The grok library is created by logstash developer Jordan Sissel and is available for other tools to be use. Grok itself is based on regex, but makes it easier to write rules because it allows to give names to regex patterns and use these names instead. To match our example from above the pattern could look like this:

```
sshd [%{NUMBER:pid}:] Accepted %{WORD:auth_method} for %
{WORD:username} from %{IPORHOST:src-ip} port %{NUMBER:src-port}
ssh2
```

A nice additional tool available for grok is grokdiscovery. This tool takes a sample log message and tries to predict the pattern that could be used to normalize this message. Of course this is not always directly usable, but speeds up the creation of rules with grok.

3.4.5 Heka

Mozillas Heka includes its own transformation. It is based on regex, but it is easier to read, because it includes the variable name inside the regex. The following is an example for parsing the Apache combined log file format. Some lines where deleted here that would have defined the type of the fields.

```
match_regex = '/^(?P<RemoteIP>\S+) \S+ \S+ \[(?P<Timestamp>[^\]]
+)\] "(?P<Method>[A-Z]+) (?P<Url>[^\s]+)[^"]*" (?P<StatusCode>\d+)
(?P<RequestSize>\d+) "(?P<Referer>[^"]*)" "(?P<Browser>[^"]*)"/'

timestamplayout = "02/Jan/2006:15:04:05 -0700"
```

3.4.6 Filter_regex

The Node-Logstash tool uses a pure regex based normalization. The configuration is a lot more error prone, as it is visible in this example:

```
{
```

```
  "regex": "^<(\\S+)>(\\S+\\s+\\S+\\s+\\d+:\\d+:\\d+) (\\S+)
([^:\\[]+)\\[?(\\d*)\\]?:\\s+$Accepted \ (gssapi(-with-mic|-
keyex)?|rsa|dsa|password|publickey|keyboard-interactive/pam) \ for
[^[:space:]]+ from [^[:space:]]+ port [0-9]+( (ssh|ssh2))$",
"fields":"syslog_priority,timestamp,@source_host,syslog_program,sy
slog_pid,auth_method,username,src-ip,src-port",
  "numerical_fields": "syslog_pid","src-port"
  "date_format": "MMM DD HH:mm:ss Z"
}
```

3.4.7 nxlog

Nxlog also offers some limeted Transformation between formats. It can convert for example a Windows Eventlog to a JSON or GELF message, but can not convert unstructured log format into structured ones.

3.5 Storage

The traditional way to store log messages is a log file. This may be bad for searches, but there are some advantages to it. In most cases some kind of database system should be used.

3.5.1 Log files

Traditional log files may feel antiquated, but they have the big advantage that they are readable in the future. 10 or even 30 year old log files can be read today, if the physical medium is still readable. New features make log files even more interesting.

Since rsyslog version 7.4 it is possible to create signed log messages with the help of guardtime [Gerhards2013]. This uses a Keyless Signature Infrastructure and a hash-tree or Merkel-tree to put multiple small log messages together and then uses linked timestamps to make it tamperproof. The Cryptographic information is shown in [Gerhards2013-2] and at www.openksi.org. Rsyslog's approach is targeted to be used when writing to a log file. It is not possible to be used before it is send to the central log server. This is a design decision that comes from the idea that not all log messages that are send to a central server will be saved.

Systemd's journal also has a signing feature. It uses Forward Secure Sealing (FSS) to achieve a similar objective. Instead of the Keyless Signature Infrastructure it uses a cryptokey that is displayed as ASCII and QR-code during creation. This can be scanned and be used to check if the log file has been altered. The log files are stored locally and can be deleted by an attacker. This problem is acknowledged by the author in [Poettering2012].

3.5.2 SQL

The idea to use SQL to save syslog data is not new. Both rsyslog and syslog-ng have been supporting SQL databases for a long time. The problem is that you cannot really split up the unstructured log message, so the table structure of such a database is quite simple. Only the syslog structure log host, date, facility and priority can be stored and the message is a long string field. There are some possibilities to speed up searches via full text search extensions like Sphinx.

Storing structured log data in a SQL Database is not easier, because there are too many different fields possible. The fixed schema of SQL is not flexible enough to be used for that.

The tools ELSA (see page 35) and rsyslog's LogAnalyzer (see page 38) are using MYSQL (see page 41) to store the log messages in an SQL Database. Rsyslog and syslog-ng as well as other tools, are supporting other SQL dialects as well. Some with the help of the DBI library, some with native support.

3.5.3 NoSQL

The problem with the schema and structured log files was one of the reasons to move to a NoSQL database, primarily a document store. The document store is a NoSQL database like MongoDB and Elasticsearch and is storing documents in JSON, XML or other data formats. These documents can be indexed and replicated to speed up searches and make the system more reliable.

There are primarily two NoSQL databases used with log management, MongoDB (see page 41) and Elasticsearch (see page 42).

3.5.4 Compare Storage

Program	**mongodb**	**hadoop**	**elasticsearch**	**SQL (DBI or native)**
logstash	w		rw(logstash format)	
graylog2			rw (graylog format)	
rsyslog	w	w	w(logstash format)	N/DBI
syslog-ng	w			DBI
node-logstash			w(logstash format)	
nxlog				DBI
Heka			w	
fluentd	w	w	w(logstash format)	DBI
flume		w	w(logstash format)	

Table 5: Storage Overview

3.6 Analysis

Simply storing the normalized log data is not enough, to get some more usage from the log files the data in it has to be analyzed. The main reason to analyze the log data is to detect problems, attacks and to correlate events. Some events need only to be noticed if a lot of them occur. One logon error is nothing to worry about, 1'000 logon errors are not normal and should be checked. A 404 error on a webpage is ok, 1'000 per second is not ok. This kind of analysis should be done automatically, based on a written rule set.

3.6.1 nxlog

Nxlog has an analyzing functionality. It has a special module called event correlator (pm_evcor), but it also supports simple statistical counters like RATE, COUNT, AVG or the change rate of the RATE called GRAD. This makes it possible to create some simple analysis, but there are some problems with this as written in the nxlog documentation [nxlog-var-warning].

With the event correlator module it is possible to create rules to ignore messages that arrive too often, to avoid being flooded by warnings. It offers the command "pairs", that looks for events that have a matching pair, the login and logout message of a user is a good example of such a pair. The command "absent" will search for broken pairs, without the second part arriving inside a certain timeframe.

The following example will send a warning if the field "Message" containing "login failure" is detected 3 times in 60 seconds.

```
<Thresholded>
          Condition  $Message =~ /^login failure/
          Threshold  3
          Interval   60
          Exec    $raw_event = "login guessing in progress";
</Thresholded>
```

3.6.2 SEC

The Simple Event Correlator (SEC) is a universal event processing tool, that cannot only be used for log files but for fraud detection and other event correlation as well. SEC is written in perl and uses regex to correlate the messages.

As written on the [SEC] webpage: "SEC reads lines from files, named pipes, or standard input, matches the lines with patterns (like regular expressions or Perl subroutines) for recognizing input events, and correlates events according to the rules in its configuration file(s). SEC can produce output by executing external programs (e.g., *snmptrap* or *mail*), by writing to files, by sending data to TCP and UDP based servers, by calling precompiled Perl subroutines, etc."

The following example from [Vaarandi2012] shows a rule that checks ssh, apache and iptables/netfilter for attacks and sends a mail when an attack is detected:

```
type=EventGroup3
ptype=RegExp
pattern=sshd\[\d+\]: Failed \S+ for (?:invalid user )?\S+ from
([\d.]+) port \d+ ssh2
thresh=3
ptype2=RegExp
pattern2=^([\d.]+) \S+ \S+ \[[^]]+\] “[^”]+ HTTP\/[\d.]+” 4\d+ \d+
thresh2=1
ptype3=RegExp
pattern3=kernel: IN=\S+ OUT= MAC=\S+ SRC=([\d.]+)
thresh3=5
desc=Repeated probing from $1
action=pipe ‘Repeated probing from host $1’ /bin/mail
root@localhost
window=120
```

3.6.3 Sagan

Sagan is a real-time log analysis & correlation tool and is written in multithreaded C. Sagan rules look similar to the rules of the Snort Intrusion Detection System (IDS) to simplify rule management with oinkmaster and similar tools. The log messages have to be delivered in a special pipe (|) separated format via a FiFo socket. As an output it can write directly to a log file or uses barnyard2 to write to a SQL database. This is the same mechanisms that is used by snort. It uses liblognorm for normalization and its own rules that are similar to snort rules.

The following rules will create a warning if more than 5 authentication failures can be detected inside a 300 second timeframe:

```
drop tcp $EXTERNAL_NET any -> $HOME_NET $SSH_PORT (msg:"[OPENSSH]
PAM Authentication failure - Brute force [5/5]"; content:
"Authentication failure"; classtype: unsuccessful-user; reference:
url,wiki.quadrantsec.com/bin/view/Main/5000015; normalize:
openssh; program: sshd; after: track by_src, count 5, seconds 300;
threshold: type limit, track by_src, count 5, seconds 300; fwsam:
src, 1 day; sid: 5000015; rev:5;)
```

3.6.4 Logstash and metrics

Logstash can be used for some analysis jobs. There is a metric plugin that can create rates calcuations for 1, 5, and 15 minutes, as well as min, max, stddev and avg. The problem is that there is no way to use it directly, you can only forward it via JSON to another tool, like a grapher as explained in the next chapter. Also missing is the possibility to check if a special user has been mistyping his password a certain amount of times in the last couple of minutes.

3.6.5 Graylog2

Graylog2 has the possibility to put messages which are selected by a search query into a message stream. When a certain amount of messages arrive in a stream, it can trigger an alarm and can send mails, using jabber or call an external plugin. It can also forward all messages from a stream to an output plugin like an external paging service, but it also misses checks against things like guessing passwords. Streams only work with new messages that arrive, not with messages that are already stored in the elasticsearch database.

3.7 Visual output

All the collected, normalized and analyzed log files can be stored, but without a visual output no one will notice. There are multiple web applications that can show different aspects of the log messages, most are integrated into a log tool, like ocotpussy, graylog2 or ELSA (see chapter Multi purpose tools on page 31).

There are two kibana projects that are working with logstash, but are developed separately. Kibana 3 is even usable with other tools like graylog, as long as it uses timestamps and elasticsearch as storage. More about the different web front ends in chapter Webpage on page 33.

If you only want some graphs to be added to an existing web site, special graphing tools are available. These graphing tools can be found in chapter Graphs on page 40.

4 Tools

Most of the tools used for structured log file analysis offer multiple components in one program. In the last chapter the different parts that are necessary for the creation were introduced. This chapter shows the different tools with all parts that are built into the tools. This chapter begins with the multi purpose tools, then the different outputs, then storage, transport, shipper/collector and finally the analysis tools.

4.1 Multi purpose tools

Some tools can be used for a wide range of purposes, some others are only created for one specific purpose. This section begins with the multi purpose tools.

4.1.1 Syslog-ng

Syslog-ng is a syslog server created by Balabit.com a Hungary based company. Syslog-ng exists in two versions: an Open Source Edition (OSE) and a Premium Edition. The latter is only available for paying customers with support and it is not Open Source. Because of this, syslog-ng was almost removed from this thesis, but it is used quite extensively and the missing features are not big enough to warrant the removal. The features missing in the OSE version include: Handle Multiline messages, encrypted log files, reliable log transfer, client-side failover and buffering log messages persistently to hard disc in case the destination becomes unreachable. In this thesis whenever syslog-ng is written, it is about the OSE version.

Syslog-ng was the default syslog in SuSE Enterprise Linux (SLES) and OpenSuSE, but it is being replaced by rsyslog [SLES2013]. It is unknown if that was because of the OpenCore nature of the development, or to be in sync with other distributions like Debian and Red Hat.

Syslog-ng cannot only be used to collect syslog messages, but also as a shipper reading the files directly. The support for reading multiple files with wildcards is only supported by the closed source Premium Edition. The same goes for the handling of missing syslog servers. When the target server is not available, the log messages are not stored, but lost.

There is a huge amount of plugins available inside syslog-ng including writing to SQL Databases, MongoDB and AMQP. With the help of an AMQP Cluster it is possible to make sure that syslog-ng does not lose messages when a server is down. Syslog-ng does support encryption out of the box.

Syslog-ng has its own normalization tool called Pattern-DB with a huge amount of predefined rules. This ruleset is very actively maintained [Czanik2013]. See chapter 3.4.1 on page25.

The documentation of syslog-ng is very good, well structured and extensive.

The OpenCore nature of syslog-ng is a big problem, but the huge and actively maintained Pattern-DB is something that is not available anywhere else.

4.1.2 Rsyslog

Rsyslog started as a replacement for the traditional syslog and as an opponent for the existing syslog-ng. As developer Rainer Gerhards wrote in [Gerhards2007] it was developed to be a real Free Software and Open Source alternative, because syslog-ng has become a dual-licensed open core product. Rsyslog is completely Open Source and Free Software and support is available from the German company of the original author named Adiscon.com.

Rsyslog is the default syslog server for Fedora, OpenSUSE, Debian and RedHat Enterprise Linux and available in every major linux distribution. It supports a large amount of input and output plugins as shown in Illustration 3.

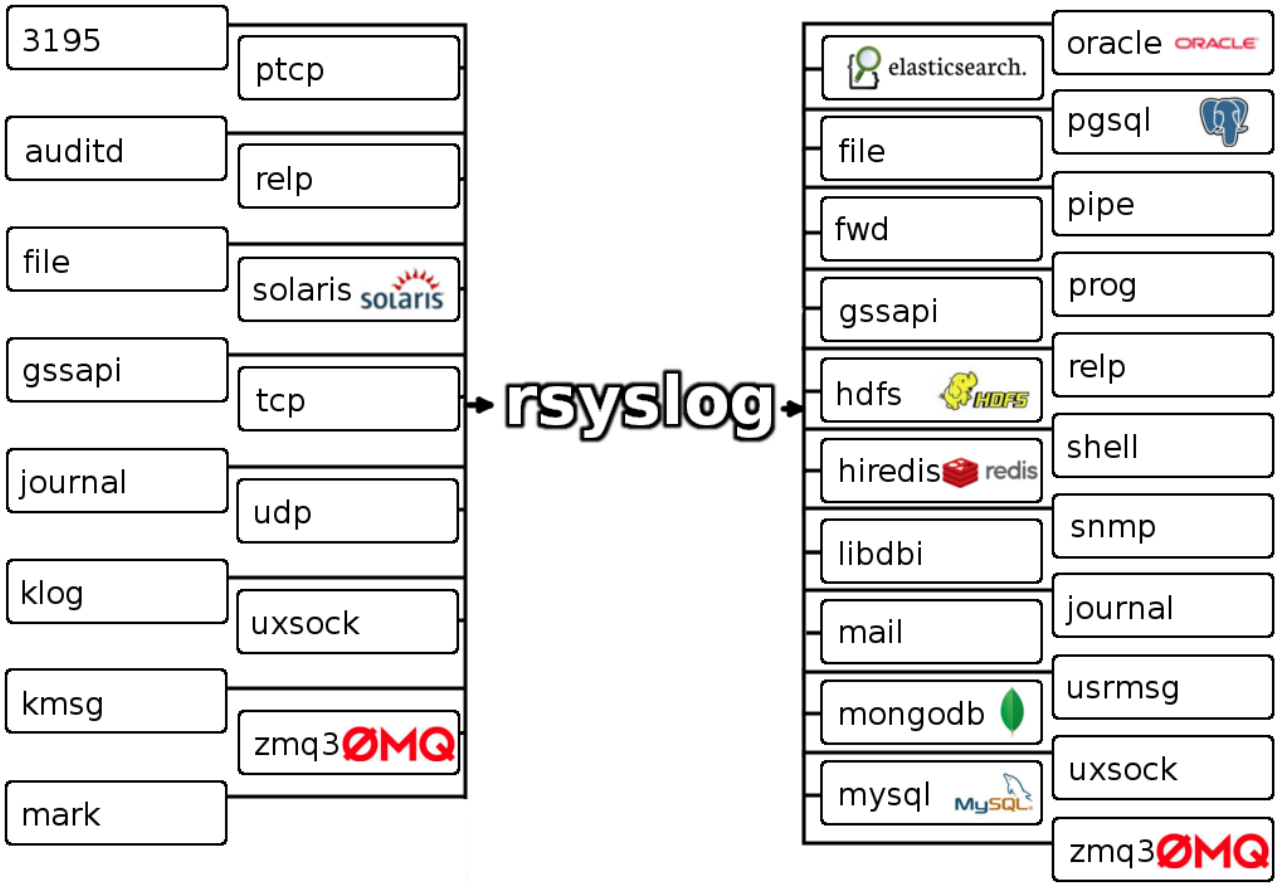

Illustration 3: rsyslog in/out plugins

The possibility to write to mysql makes it possible to use the Log Analyzer tools from page 38.

One of the unique features of rsyslog is the repl plugin, that makes it possible to make sure that syslog messages are really received by the server. This only works with rsyslog, because it is not a standard. Together with the disc based queue it is very easy to make sure no message gets lost.

This can also be done via the Ømq plugin. Both these reliable mechanisms suffer from a lack of encryption. Only the encryption of the normal syslog traffic is available. The development of encrypted repl is under way but not available yet.

Syslog offers a normalization library named "Liblognorm" that is described on page 25.

The documentation of rsyslog is strange, because a lot of interesting features are only explained in blog posts from the main author.

4.1.3 Graylog2

Graylog2 is a complete, Free Software and Open Source log management solution, created by Lennart Koopmann and is supported by torch.sh – a German based company. It stores the data in an elasticsearch cluster and the statistics and graph data in a MongoDB. The log messages can be send via syslog in an unstructured way or in the own structured format named GELF, see page 14.

Graylog2 supports the creation of multiple graylog2 instances, writing to the same elasticsearch cluster. This allows to create a fail over setup. Together with the AMQP support in graylog2 it is possible to have different graylog2 nodes connecting to the same AMQP broker infrastructure. In this setup graylog2 nodes will automatically distribute the messages to share the load.

The biggest problem of graylog2 is the fixed requirement to a specific elasticsearch version. This happens because graylog2 adds its own elasticsearch node into the cluster. This cluster node can be configured to store data itself or to solely forward the data to other data nodes. This can create some problems, because the elasticsearch development is quite fast, and you have to use an old version to run graylog2. Another problem with elasticsearch is that it handles deletion of old log files not based on date, but only on the size of the log files. This makes it easy to manage the disc space, but it is not known how many days of log files are available.

The webpage from graylog2 is more than a simple dashboard, it allows to sort messages into streams. Streams are query results that can be used for monitoring and alerting other programs, as written on page 30. Streams can very easily be created from the webpage and be put into categories to make handling of a huge amount of streams easier. The web interface also supports adding admin messages to log entries, if a special problem is known. It is possible to write a regular expression and for every log entry that fits this expression, an automatic message is added to the web page.

Graylog2 offers to remove sensitive information like passwords. Streams can also trigger alarms, send mails, jabber messages or an external plugin. All messages of a stream can also be forwarded to an output plugin. To round it up graylog2 allows to put machines into host groups.

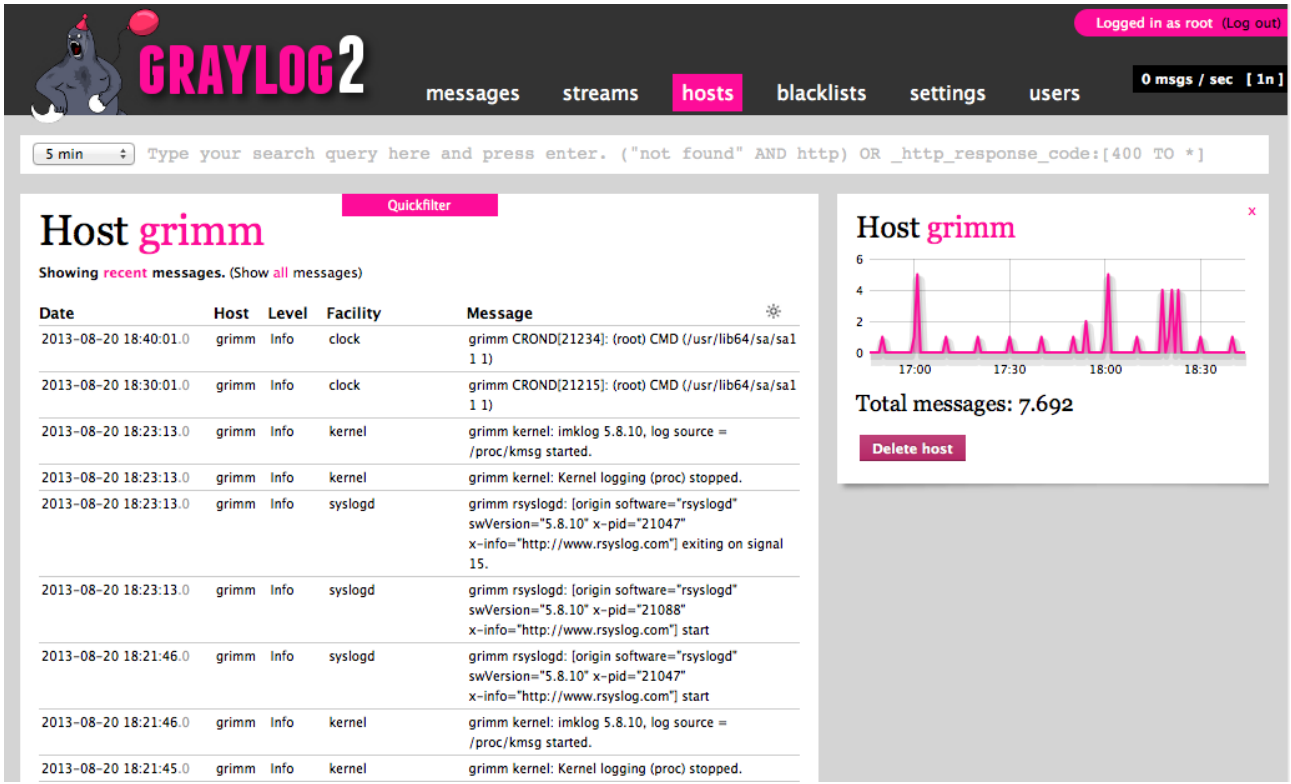

Illustration 4: Graylog2 web page

4.1.4 Logstash

Logstash is the "swiss army knife" of log management. Containing everything from transports, to collecting local sources, to normalizing log messages, to storing data in elasticsearch up to a webpage to query the data from elasticsearch.

Logstash development was started in 2009 by Pete Fritchman and Jordan Sissel and has a huge number of input and output plugins as seen on page 57, as well as filter plugins. The filter plugins includes plugins that can be used for: anomymization, convert to the GELF, JSON, KV, XML and other formats, resolve ip address into geo coordinates, grok as described in chapter Grok on page 26, merge multilines (like stack traces) into one message, split one message into two, translate number into text (like error codes into error message) or resolve IP addresses into hostnames.

The documentation is very good, and includes a very helpful introduction. If more information is required a logstash book written by James Turnbull is also available [Turnbull2013].

Logstash's own webpage is very limited in its usage. It can be used to query elasticsearch, but it is missing a lot of other features offered by the competitors. The big advantage is the simple installation. When logstash is already running, a simple command line with the parameter "web" starts the web server.

Several reliable transports are available, like AMQP and syslog with RELP. It also supports encrypted syslog, but only without RELP. With tools like lumberjack it supports both reliable and encrypted transports as well.

As the logstash webpage is very limited, instead the tools Kibana 2 or Kibana 3 are often used.

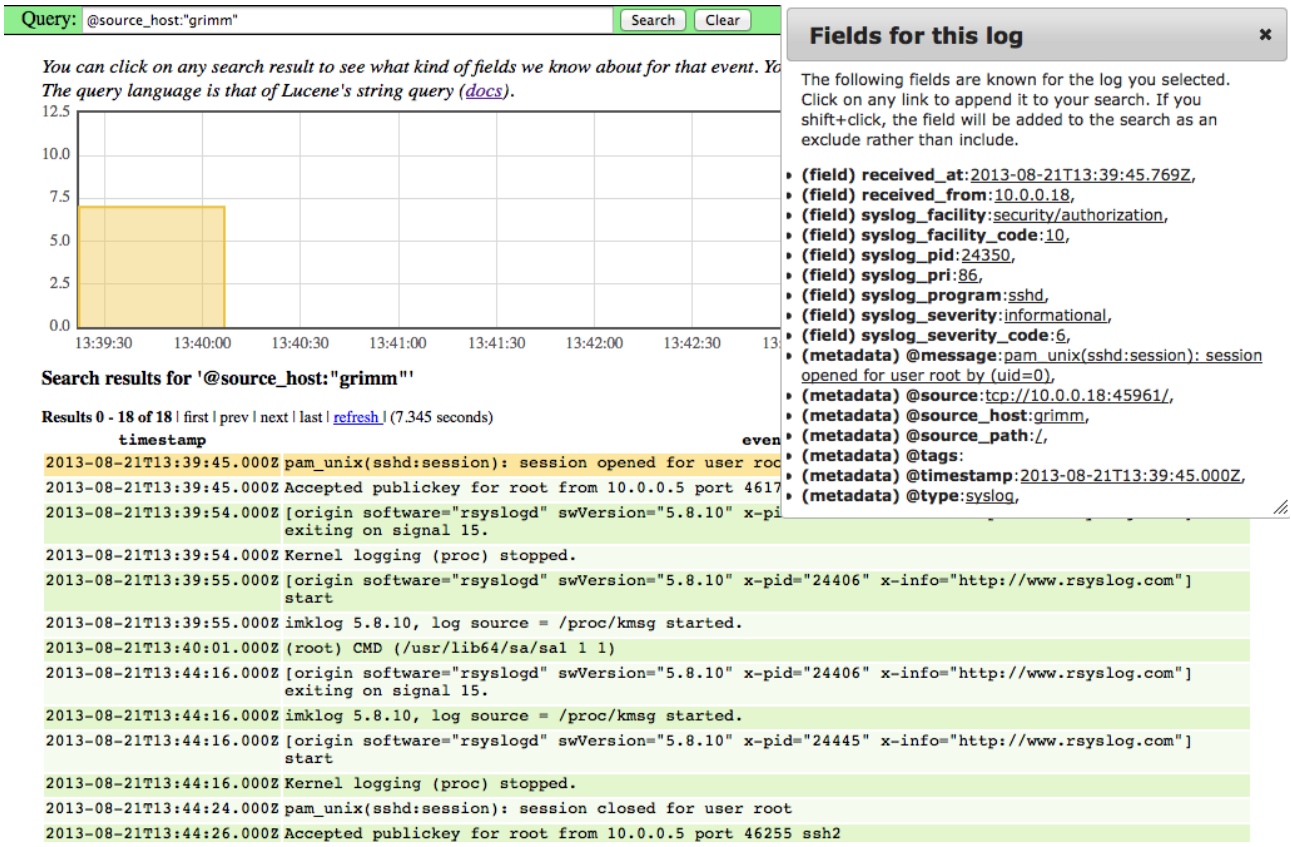

Illustration 5: Logstash web page

4.1.5 Node-Logstash

Node-logstash is a reimplementation of logstash written in Javascript, based on node.js and developed by Bertrand Paquet. It also uses elasticsearch, but it is not limited to a specific version. Node-logstash has not yet the same amount of plugins as the original. The grok plugin is missing, and a replacement plugin called filter_regex is working with a regex rule base. This is shown in chapter Node-Logstash on page 26.

The Filter plugins are: add_source_host, add_timestamp, compute_date_field, compute_field, grep, json_field, multiline, mutate_replace, reverse_dns, split, syslog_pri and regex. So a lot of the functionalities are missing. A reliable transport is available with redis, but there is no encryption available whatsoever.

The project started very recently in July 2012, but is very actively developed.

4.1.6 ELSA

The Enterprise Log and Search Application is a combination of syslog-ng, mysql and sphinx. It is written primarily in perl by Martin Holste. He started the project in 2011.

As written in [Holste2011] the development was driven by the need to create a logging server that could be queried very fast. According to the [ELSA-UserGuide], it is using the Pattern-DB from syslog-ng for normalization, and forwards the normalized log messages to the perl programs. These are sending the messages via bulk load to the mysql database. The sphinx server is indexing the new data every few hours to gain speed and to work with larger chunks of data. After a defined amount of time the data is moved from the MyISAM table to a table from type ARCHIVE. The query language is based on the google query language, which makes it very easy to use, but does not support wildcard searches, that most other web front ends offer.

Google also provides a lot of images, Javascript and css files, that make it impossible to use it without internet access. For a tool with such security and privacy related data, it is surprising that it gets most of the files from google.

The installation is a little unusual, where most tools tell you to install a list of requirements and then you have to install the package, ELSA only offers an install script that does the installation. Separated into "node" and "web" it installs a lot of programs like mysql, apache, gcc, header files for different development packages and a huge amount of cpan modules. It also downloads syslog-ng and sphinx and compiles it. It also downloads its own source and cpanm from the web. The nice thing is at the end it runs a self-check that puts some messages into syslog and tests if these are correctly stored and indexed.

It is possible to create multiple ELSA nodes, but these nodes are not replicated, but instead every node runs independently with its own messages, databases and index server. The webpage will send the necessary queries to all ELSA nodes in the cluster, so it looks like all information is stored in one datasource. This has the advantage that every node is robust and if a node is missing the information of this node is missing too, but everything else it unaffected.

ELSA also offers email alerting, a plugin architecture and host checks that informs about hosts that are not sending messages anymore. A nice feature is the possibility of defining log classes and defining which user can access which message. This makes it possible to define that a web developer can access the web logs, but not the ssh or audit logs. It also offers dashboards for a better overview over different kind of log messages.

The documentation is quite extensive, but to ask questions it is necessary to have a google account.

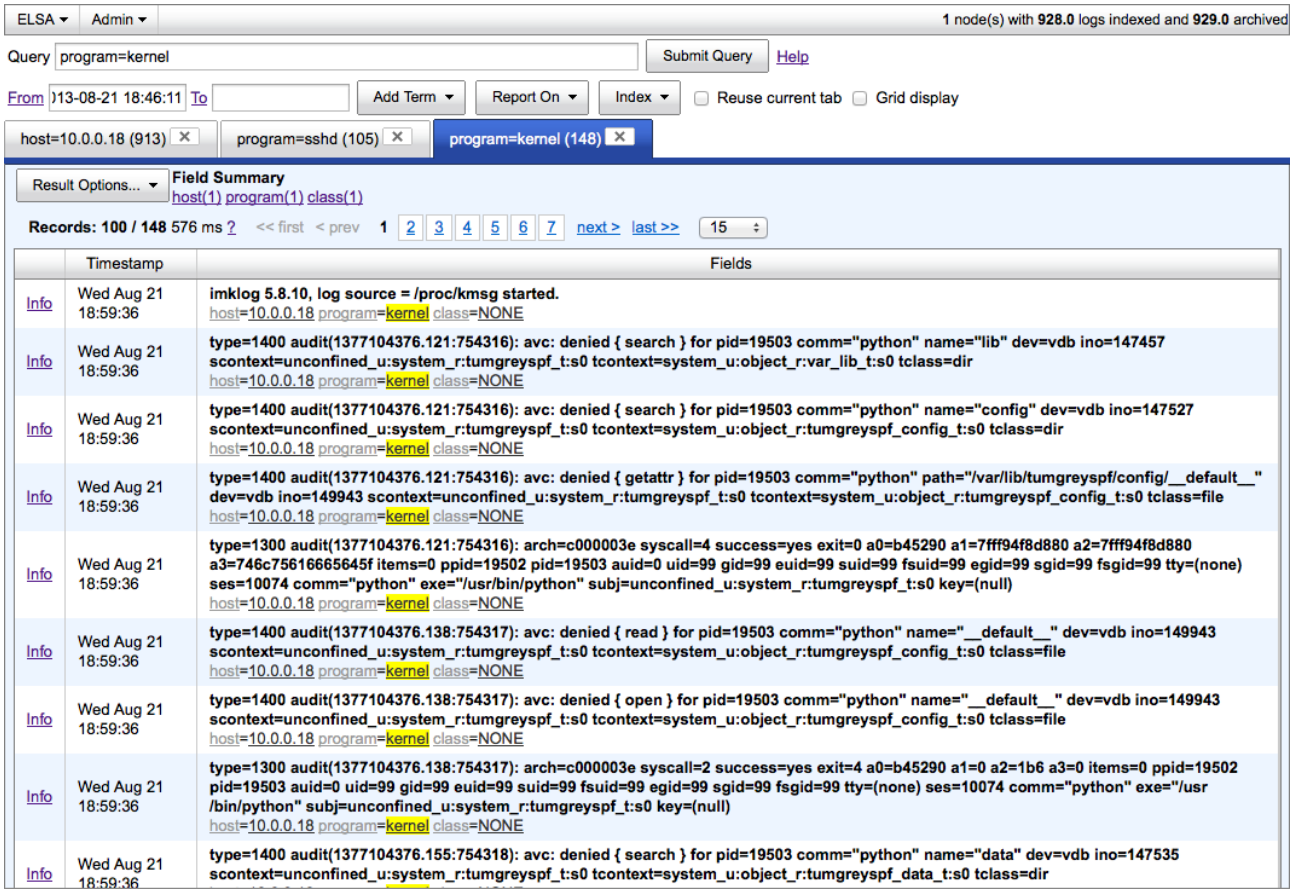

Illustration 6: ELSA web page

4.1.7 octopussy

Octopussy or 8pussy is a quite old project and was started in 2005 by Sebastien Thebert. It is written in perl and is available as a Debian archive or as source code. It brings its own normalization library as shown on page 25. It uses rsyslog to accept the messages and sends them via a fifo into the ocotpussy dispatcher. The dispatcher sends the messages to the parser (for the normalization). For every host that sends messages to octopussy a new parser is started. This can be a problem with larger setups, because every octo_parser has a resident set size of 24 Mebibytes. With thousands of hosts this can be a problem.

On the webpage it can be defined which services are running on the host and automatically the ruleset for this service is added. A huge amount of predefined log formats or services are available, not only Linux and Windows Services, but also MacOS, Netscreen, Ironport, F5 and Cisco. It is also possible to add ones own services and rules. There is even a wizard that shows all unidentified log messages and helps with the creation of rules for these log messages. There is LDAP authentication available, as well as alerts, reporting and rrd-based graphics.

The big problem is the data storage. There is a mysql database required, but the logs are stored based on the detected service in a compressed cleartext file. These files are stored in a date based file hierarchy with one log file for every minute of the day, up to 1440 files per directory. This can limit the search speed. A fast search is possible, but only if the search is limited to a specific service, because only these logs have to be uncompressed and searched.

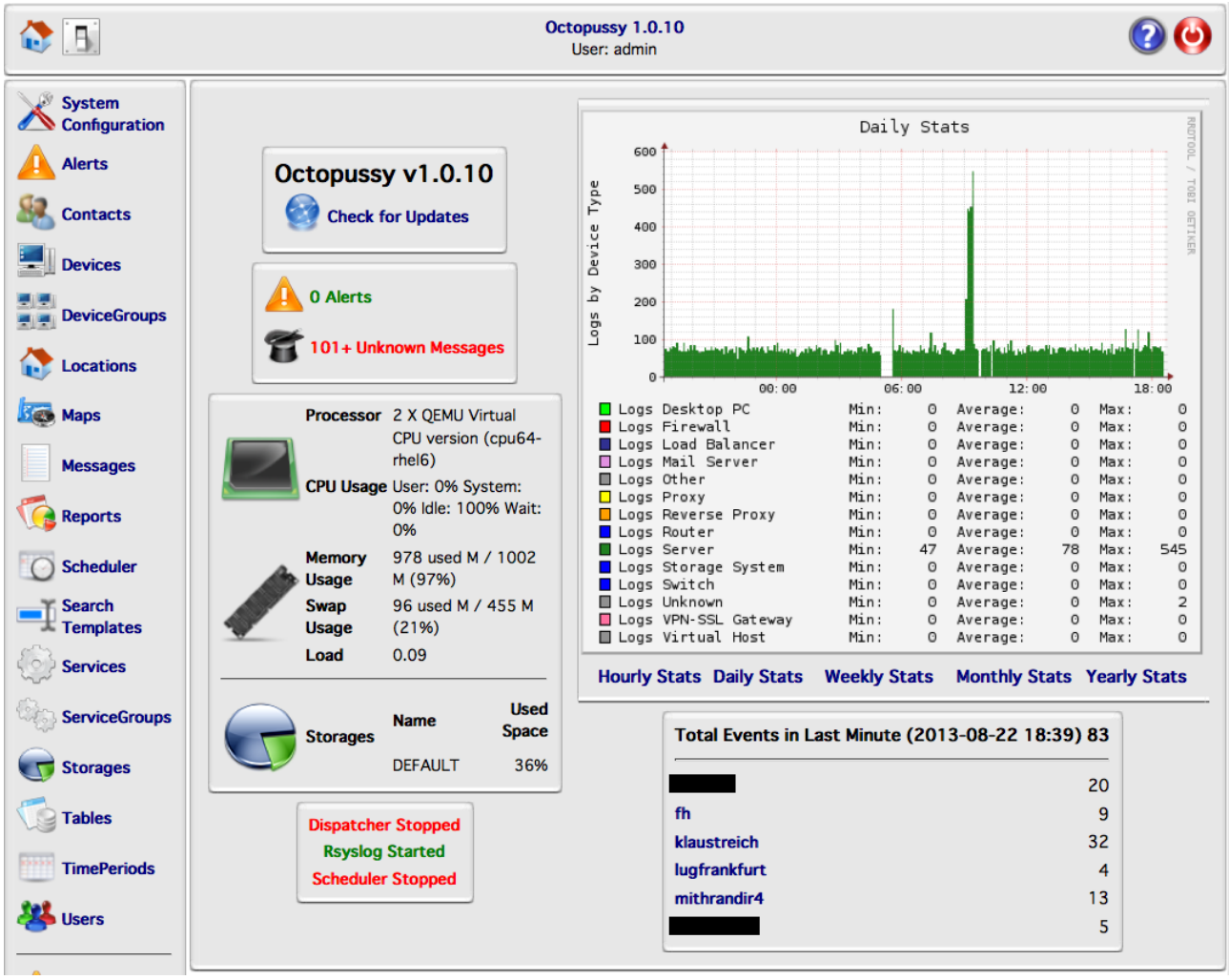

Illustration 7: Octopussy home page

4.1.8 nxlog

Nxlog is a very universal log collector and shipper, together with some analyzing and normalization capabilities. It is written in multithreaded C and is created and supported by the Hungary based company Nxsec. The code is only released on SourceForge as a tar.gz archive and no source code repository like SVN or GIT is available. Nxlog is OpenCore software; some feature are only available with the "Enterprise" version, this includes better Event correlation, http REST api, snmp input and a remote windows event collection. But the important features are available in the Open Source version.

The architecture is based on plugins, but these are called modules here. The modules are separated into extension modules that add support for message formats like syslog, gelf, JSON or multi line message parser to handle Java stack trace. There are input and output modules as well as process modules with support for memory and disk buffers for bridging server gaps. It also offers event correlation and message de-duplication. Encryption is available as an input and output module.

The analyzing and event correlation is described on page 28.

The [nxlog] documentation is quite complete and includes a lot of examples.

4.1.9 Heka

In April 2013 the Service Team of the Mozilla Foundation announced the first public release of Heka on their page [HekaIntro]. Heka is primarily designed as a shipper, but has some support for normalization. It is written in go, but can be extended in the language lua as well. It uses RabbitMQ as the primary transport, but does not support the TLS encryption of AMQP. It can write directly to elasticsearch since version 0.3 released in July 2013. Heka is a very young product, but with the support of the Mozilla Foundation it could become very interesting in the coming months.

Heka has some log normalization features as shown on page 26.

The Documentation is surprisingly thorough for such a young project.

4.2 Output

Some webpages and graphics generators are tool independent. These are shown here.

4.2.1 Webpage

The Webpages shown here are not belonging to a special tool or framework, but are developed independently.

4.2.1.1 LogAnalyzer

The LogAnalyzer is created by the same developer as rsyslog and is using rsyslog to write syslog data into a MySQL database and this data is shown via this webpage. Strictly speaking this tool should not be shown here, because it does not use structured log data, instead it only uses the semi-structured log data from syslog. But there is a possibility to extend LogAnalyzer to handle structured data as described on the rsyslog webpage [Gerhards2011-2].

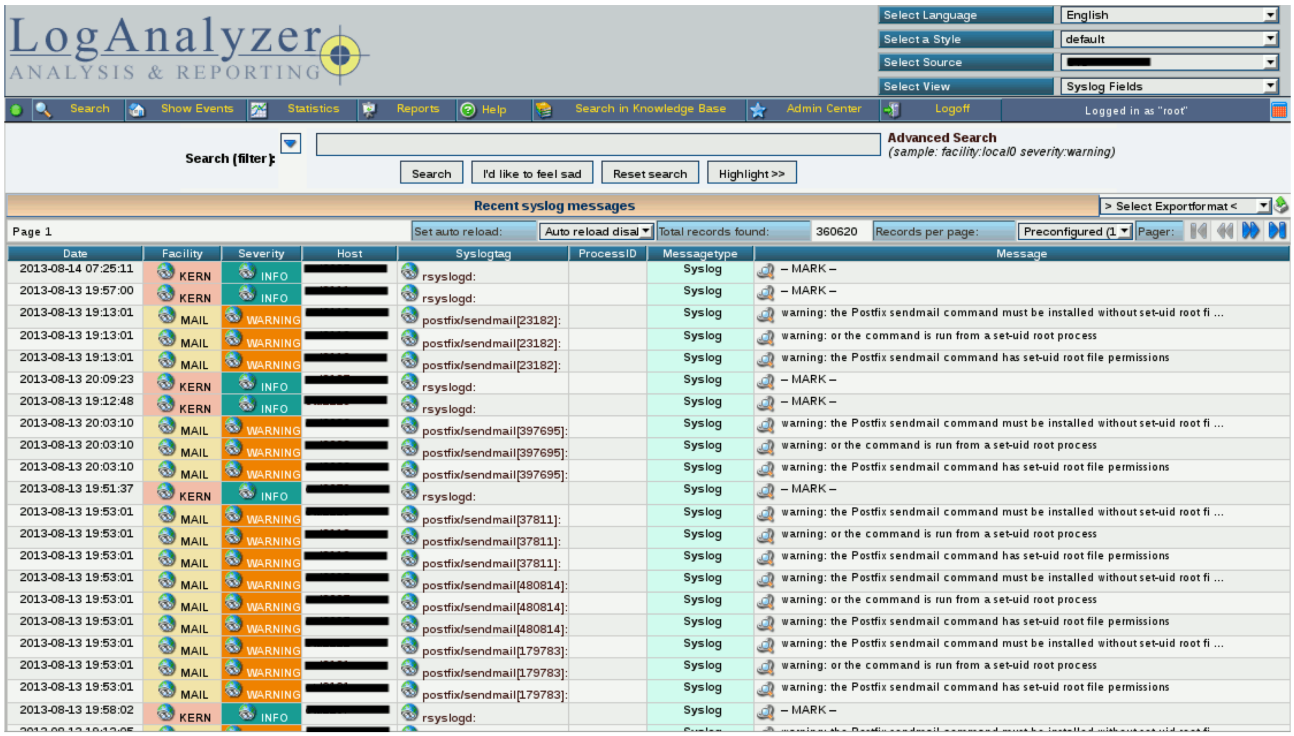

Illustration 8: LogAnalyzer web page

4.2.1.2 Kibana 2

Kibana 2 is a web front end for accessing the data written to elasticsearch by logstash. Kibana 2 is based on Ruby and needs a lot of Ruby gems. It is suggested that these are installed with the help of the Ruby gem bundler. Kibana 2 is offering interactive graphs and can show trends and distribution of fields in the log data. It even can create dashboards or rss feeds based on lucene queries.

Kibana 1 was based on php, but is long abandoned.

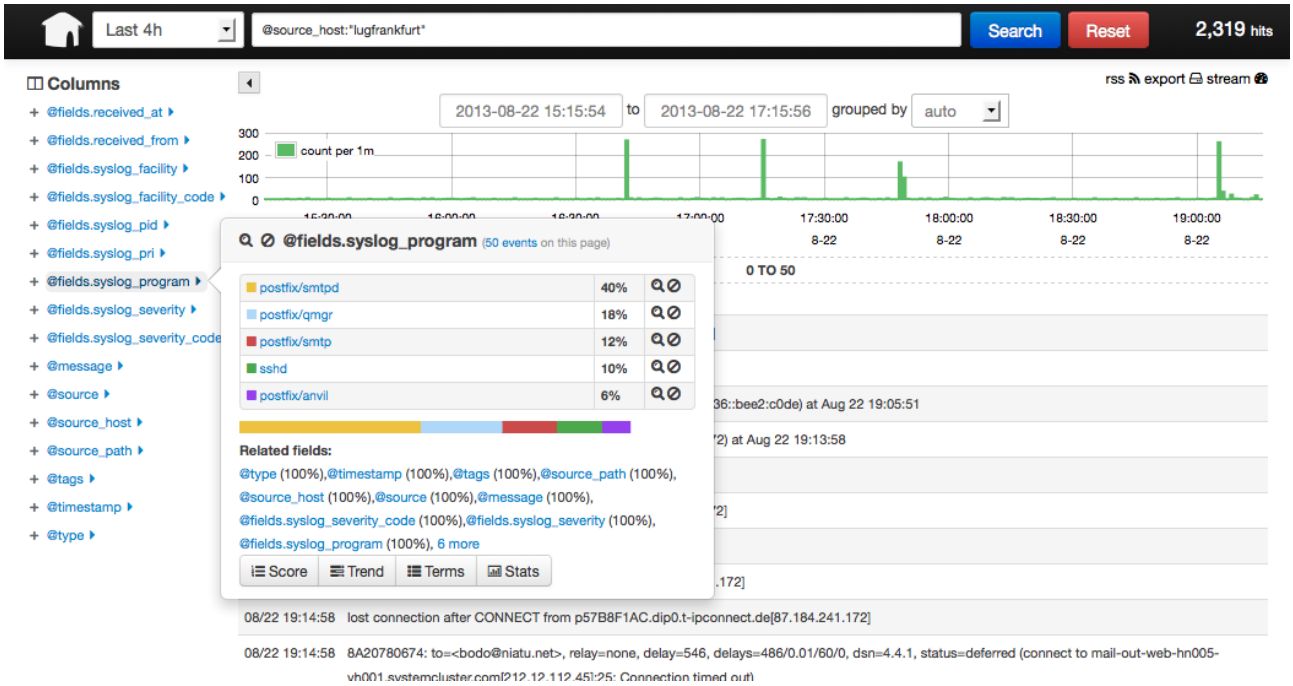

Illustration 9: Kibana 2 web page

4.2.1.3 Kibana 3

Kibana 3 is a new version of Kibana written completely in HTML5 and Javascript by the elasticsearch team. The code running in the web browser talks directly with the elasticsearch server. It is not yet considered stable, but is working quite well already. The difference between kibana 2 and 3 is not only the programming language, but the way kibana 3 works without a fixed format in elasticsearch. It is possible to use it on other data as well, as long as there is a time field. It is even possible to use graylog2 format with kibana 3. Because of the use of HTML5 it does not need anything special on the server side, but it needs a modern web browser on the client side and therefore it does not work with an older version of "Internet Explorer" which are used at too many companies.

Kibana 3 is more like a web based dashboard creation tool, than a simple dashboard. It ships with an example for logstash, but it can be very easily extended and rewritten without a single line of code.

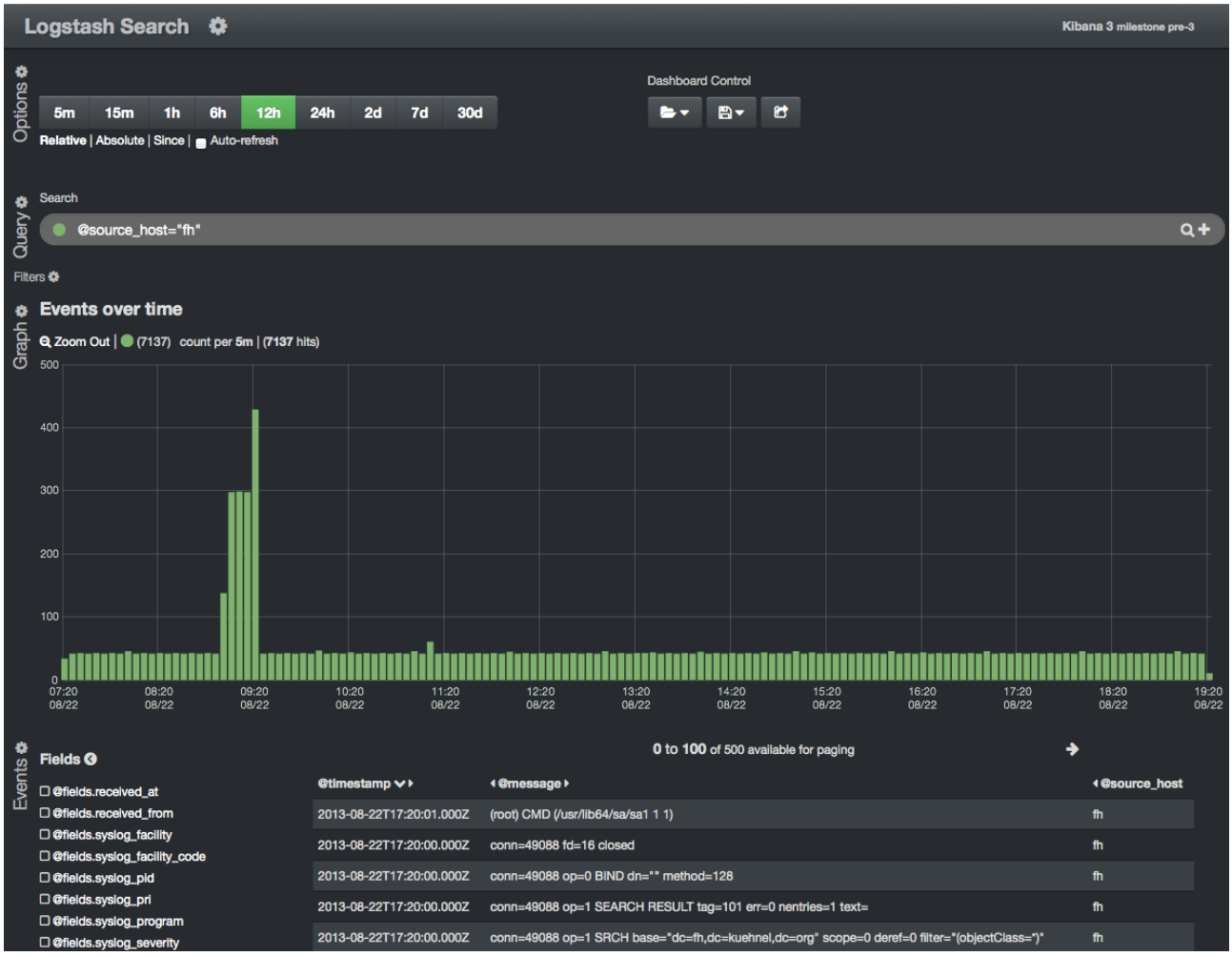

Illustration 10: Kibana 3 web page

4.2.2 Graphs

The different web front ends can create some simple graphs, but if the graphs should be used in some other website like monitoring, a special graph tool is needed.

4.2.2.1 StatsD

StatsD was created by Etsy to follow their religion of the "Church of Graphs. If it moves, we track it." as stated on [Malpass2011]. StatsD is a simple Event Tracking system written in Javascript based on Node.js. It receives the status changes via an UDP socket. A new counter does not need to be created, simply start adding data to a new counter and graphs will automatically be created. For very frequently hit counters it is possible to send only every ten or every 100 events to StatsD and it will be correctly stored in the counter. StatsD does not create graphs, but sends the data normally on to graphite to generate the graphs.

4.2.2.2 Graphite

Graphite is used quite often together with StatsD, but can be used without it. Graphite is written in Python and uses the Twisted and Django framework. Internally it uses whisper as a database for time-series data (similar to rrd), carbon is the data point receiver and a web application is available

to display the graphs, also called metric. The internal architecture is explained in chapter 7 of [OSArch2012]. Every graph or metric has a path that is used to specify the graph and can help organize it as well, like *company.websites.logging.auth.user.error*.

The graphite messages are send in a format like this:

```
path_to_graph value unixtimestamp
company.websites.logging.auth.user.error 1 1375946427
```

The graphite server takes this information and stores it aggregated in the whisper database, based on the configuration for this tree of graphs and generates the graphs to be used by a webpage.

4.2.2.3 Fnordmetric

Fnordmetric is a collection and visualization framework for time series data. There are two backends to choose from plus a web GUI.

4.2.2.3.1 Fnordmetric Classic

Fnordmetric Classic is written in Ruby and uses a redis NoSQL database for storage. This is a Ruby framework to write Webpages with graphs, using Ruby as a Domain Specific Language with pre-build widgets to ease development. It receives data as JSON via a TCP/UDP Port or via HTTP Post.

4.2.2.3.2 Fnordmetric Enterprise

Fnordmetric Enterprise is using Scala that is run on a JVM. Fnordmetric can be used as a replacement for statsd+graphite, but the API to receive data is different. Fnordmetric Enterprise can receive data via TCP/UDP or via a http websocket. The big difference is that the name of the metric contains the metric type like mean, sum, min/max etc.

4.2.2.3.3 Fnordmetric UI

The fnordmetric UI is a HTML5 Application framework that connects to one of the two fnordmetric backends. It is possible to create a new html page or integrate it into an existing one, with only some Javascript addons including fnordmetric itself and jquery.

4.3 Storage

4.3.1.1 mysql

Mysql is the worlds most used Open Source and Free Software Relational Database. Created by the MySQL AB Company in 1995 it was bought by Sun and later by Oracle. Since Oracle acquired MySQL, the Open Source developer including the original authors were not happy with the way Oracle handled the project. They therefore created a fork named MariaDB. Both MariaDB and MySQL are quite compatible. In this thesis the term mysql refers to both databases MariaDB and MySQL. Because mysql is so famous, it will not be introduced here further.

4.3.1.2 MongoDB

MongoDB was one of the earliest NoSQL databases and one of the most used ones of its kind. MongoDB uses a binary representation of JSON called BSON. MongoDB is very old for a NoSQL system with a production ready release in 2010. MongoDB is developed and supported by the US based MongoDB, Inc. MongoDB is a document database, that can store data schema free. To create high availability setups it is supporting a master-slave configuration. This normally runs in an

asynchronous mode, so the databases are not always in sync. To split huge databases it uses sharding, where the data is distributed to different machines based on a shard key. It also supports map-reduce to distribute data and aggregation operation.

4.3.1.3 ElasticSearch

Elasticsearch (ES) is also document oriented like MongoDB, but it is designed as a pure search engine based on the Apache lucene search library instead. It provides realtime data analytics and can be distributed over multiple machines, both for load reasons as well as to improve availability. It has full text search capabilities and can store queries inside elasticsearch and execute them faster when needed. As a query language it uses the lucene syntax that includes searches in fields like this:

```
host_source:testmachine AND error
```

Elasticsearch is very easy to setup because it only needs a Java runtime, the Java jar file and a config file with the cluster name in it. A cluster is a collection of multiple elasticsearch instances that can talk with each other. To create a cluster with elasticsearch machines, simply make sure multicast is available and all nodes use the same cluster name. There is a special tutorial for using elasticsearch to store logs [Gheorghe2012].

Elasticsearch uses JSON as the document format. The JSON documents that should be stored inside ES can be put there with a http PUT request. To improve performance multiple JSON documents can be stored in one request using the bulk API. To even further improve performance a so called river plugin can be used to push documents into the ES instance. Some other type of plugins including management web front ends called "site plugins". The site plugin "elasticsearch-HEAD" is used very often and allows a fast and easy overview on the cluster and can make changes as well. There are several other site plugins available, that can give an insight into the performance and resource usage of elasticsearch.

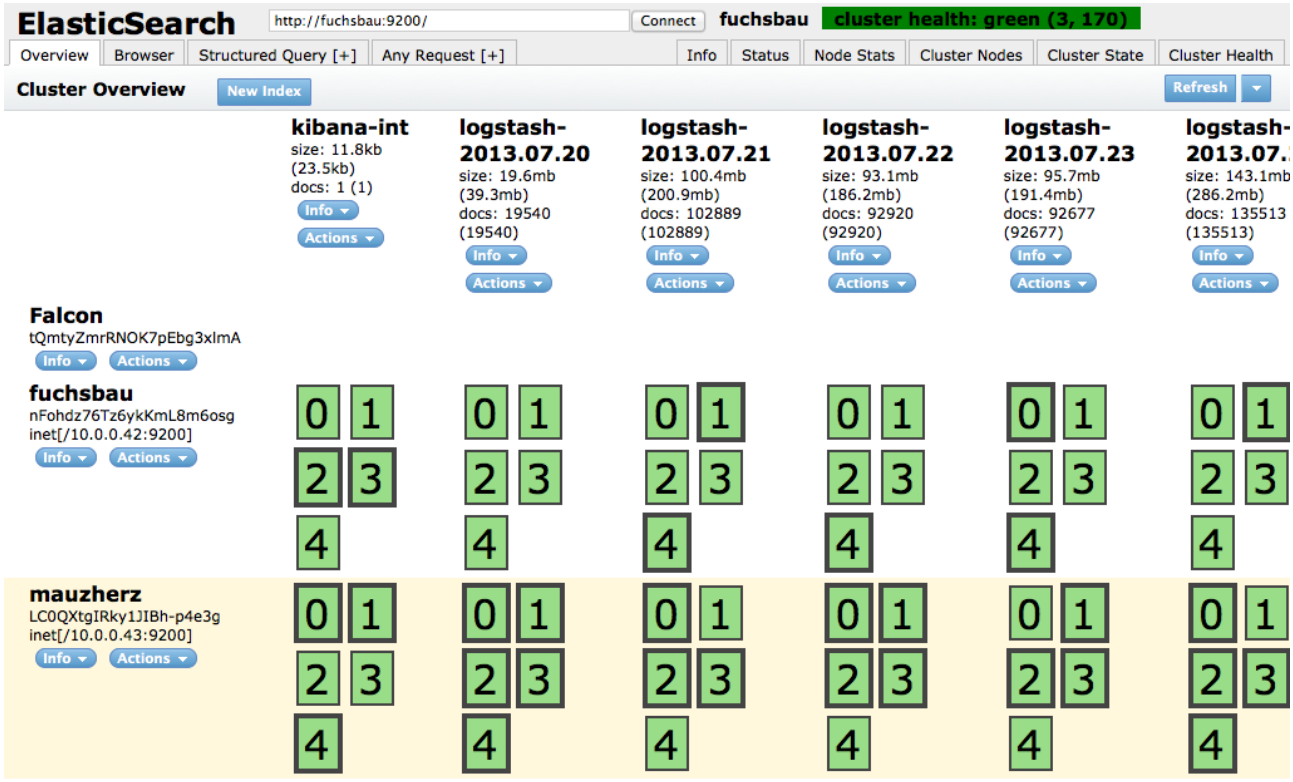

Illustration 11: Elasticsearch with HEAD plugin

Elasticsearch uses sharding and replicas to speed up access and distribute the load. Sharding is used to split up the data to be distributed to multiple machines inside a cluster. The default number of shards are five. In this configuration, if a cluster has more the 5 machines, there are some machines that do not store any data. So it is helpful to create at least as many shards as there are nodes in the cluster. These shards can also be replicated to other machines inside the cluster to work as a failover in case of a lost node. Changing the shards in an index can be very complicated, so it is easier to create the correct number of shards at creation time. Adding additional replicas on the other hand is very easy.

As Florian Gilcher wrote in [Gilcher2012] it is quite easy to create a split brain situation. This can happen when the elasticsearch nodes are distributed in two data centers and the connection is severed. In case of a split brain situation, there is no possibility to reintegrate the two sides, it is necessary to delete one side and use only the other. This will lead to loss of data, if it is not handled correctly up front.

4.4 Transports

Syslog is nice to send log messages via a network, but when it comes to reliability and security the following dedicated transport systems can offer some nice alternatives.

4.4.1 redis

[Redis] is an in-memory database that supports replication with a master-slave setup, as well as persistence with the help of snapshot and journal files. For the use as a log transport the build-in feature called channels are used to create a publish-subscribe messaging infrastructure. Together with the replication support it is possible to create a high available setup, but this is still in development and not supported yet. Redis does not support data encryption and only a password authorization as described in [redis-security]. The documentation is very good and if this is not enough there is a free book written by Karl Seguin called "a little introduction book about redis" at [Seguin2013].

4.4.2 rabbitMQ

[RabbitMQ] is an Open Source message broker that is developed by RabbitMQ Inc, a London based company, now owned by VMWare. It is written in erlang and is based on the Open Telecom Platform. Erlang is a functional programming language and famous for its possibility to restart the program in parts, without a complete restart or losing connectivity or function. RabbitMQ also contains access libraries for a lot of different programming languages.

RabbitMQ includes gateways to talk with AMQP, STOMP and MQTT. Transport encryption with TLS is available built-in, using the openssl library as written in [RabbitMQ-SSL]. RabbitMQ can be setup to cluster multiple machines in a local network into a single logical broker. This makes it possible to restart single machines without any service interruption. It is also possible to mirror queues over several machines to ensure that in case of a hardware failure no messages are lost. All this high availability is always paid with a performance penalty.

The documentation is very extensive and contains examples for all use cases.

4.4.3 ActiveMQ

ActiveMQ is also an Open Source and Free Software message broker. It is written in Java and released under the supervision of the Apache foundation. The client is available in many languages and it supports not only AMQP and STOMP, but also XMPP (former Jabber) and a RESTfull web API as written in [ActiveMQ-Features]. According to [ActiveMQ-Cluster] ActiveMQ also supports

clustering in different flavors. From failover cluster which make sure that the clients can send messages to the broker even when a node is down, to Master-Slave setup where the messages of one node are stored on a second machine to be send, if the master node goes down. ActiveMQ is supporting encryption as written in [ActiveMQ-SSL].

The ActiveMQ documentation is complete and several books are available.

4.4.4 Ømq

[Ømq] or 0MQ or zeroMQ are three different spellings for the same program. Ømq is a socket library, that can send messages to another process. This can happen inside the same process via inproc, to other processes on the same machine via IPC, or to processes on other machines with the help of TCP or Mulitcast connections. The advantage is that it does not need a special broker server running. Or as Pieter Hintjens wrote in [Ømq] - The Guide: "Ømq ... looks like an embeddable networking library but acts like a concurrency framework."

As written on page 22 Ømq does not support encryption, but the development has started to support this. Because it does not have a specialized broker it does not support clustering. It is designed for speed, not for reliability.

Ømq is created by iMatrix, a Belgium based company which provides commercial support offerings. The documentation is very good and can also be obtained as a book [Hintjens2013].

4.5 Collector/Shipper

The possibilities of the different shipper/collector can be seen on the overview on page 20.

4.5.1 Fluentd

Fluentd is a very universal shipper/collector and is developed by Sadayuki Furuhashi and written in C for the performance relevant parts and the rest in Ruby as written in the [FLUENTD-FAQ]. It is sponsored by the Company Treasure Data in California which offers a cloud based log analysis platform and uses fluentd to send the data to their cloud. It uses JSON as the native log format and can be considered a syslog of structured logs. It uses a plugin architecture, with support of output and input plugins. There are over 150 plugins available for fluentd and developing a plugin is very easy. The elasticsearch output plugin writes in the same format as logstash. It does not support any encrypted transports, but it supports reliable transport and persistent disc and memory buffer in case a target server is down. It also supports a high available setup, where multiple fluentd instances send the messages to a log aggregator running on two machines. The sender will switch over to the backup aggregator when the primary is down.

Fluentd can be installed as the fluentd Ruby gem or as a td-agent build as a deb or rpm package. The td-agent version has a slower release cycle, has a more tested release, but of course the gem version has the new features faster.

4.5.2 flume

The Apache flume project is "a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data" according to the [FlumeUserGuide]. Flume is written in Java and is supporting multiple Hadoop mechanisms. It is included in this thesis because is also supports other inputs and outputs, such as an elasticsearch output that writes like logstash. To write to elasticsearch it is necessary to add the elasticsearch and lucene-core jars into the lib directory of the flume installation, because it uses the same mechanism to use an elasticsearch node to add the data to the cluster.

The input mechanism is called source in the flume documentation and the output is called sink. Source and output are connected with channels. Flume does not support encryption outside of Hadoop, but a memory and disk based persistence in case of a server downtime is available.

The [FlumeUserGuide] is very extensive with a lot of examples.

4.5.3 awesant

Awesant is a perl based shipper that supports redis and the rabbitMQ client library. With the help of the second awesant instance running on the redis machine, it is possible to use encrypted redis.

4.5.4 beaver

Beaver is a Python based shipper that supports redis, 0mq and uses the rabbitMQ client library to access AMQP and stomp based servers. A nice feature of beaver is the support for a ssh tunnel to be created at startup. This also makes it possible to create an encrypted connection to 0mq and redis.

4.5.5 lumberjack

The logstash developer needed a shipper that was not Java based and had a very low memory and CPU requirements. To fulfill his need he created lumberjack, because there was no other system supporting that.

There are two implementations of lumberjack, a Ruby and go based one and a Ruby and c based system. Both are still receiving patches, but the go based system is much more active developed and is stored in the master branch of GIT. According to its creator [Sissel2013] it is:"encrypted, trusted, compressed, latency-resilient, and reliable transport of events". It uses OpenSSL as a base library and uses X509 certificates to check the server cert. It is possible to use client certifications as well.

Lumberjack does not support caching in case of a connection problem, but it is possible to configure multiple logstash servers that are used in case of a connection problem.

4.5.6 eventlog-to-syslog

Eventlog-to-syslog is an Open Source and Free Software Windows based Eventlog shipper. It is based on the source code from Curtis Smith (Prude University) and is now developed by Sherwin Faria (Rochester Institute of Technology). This tool supports both Windows Eventlogs formats before and after Windows 2008 and Windows Vista. It is written in C++ and sends the Eventlogs to a Syslog server. It supports the traditional syslog format via UDP and TCP and also supports server failover in case of an unreachable syslog server.

4.5.7 woodchuck

Woodchuck is a very simple Ruby based shipper that only supports redis as output, therefore no crypto support.

4.5.8 ncode/logix

Logix is a very simple Python based log shipper that is developed since 2011. It accepts udp syslog messages and sends them in gelf format to a AMQP server to be read by graylog.

4.5.9 syslog-shipper

Syslog-shipper is a very simple shipping tool written in Ruby. It can only read in multiple files and send them to a syslog server. It will add the syslog header if requested, uses TCP and support TLS encryption.

4.5.10 remote_syslog

remote_syslog is a Ruby tool, that also reads files and sends them to a remote server via syslog. It supports TLS encryption with client certificates and can detect new log files when using globs. If it is configured to look for files like /somewhere/*.log and a new log file appears it will be collected. It can even do some basic parsing functionalities.

4.5.11 systemd/journal2gelf

Journal2gelf is a very simple Python based shipper created in 2012 that takes systemd's journal messages in JSON from STDIN and sends them to a graylog2 server via GELF.

4.6 Analysis

Most analysis tools belong to a multi purpose tool. There are only two independent analysis tools.

Sagan is described on Page 30 and SEC is described on Page 29.

5 Toolchains

There are multiple ways to build a structured log analysis solution, based on the tools described in this thesis. Selecting the correct solution is not an easy task. Chapter 4 tries to give an overview on the different advantages and disadvantages of these tools.

5.1 Possible toolchains

Because of the large number of possibilities to combine the different tools it is necessary to get an overview on the toolchains which can be build. Because there are so many collectors/shippers available, this thesis will start from the webpage side to show an overview on the possible tool stacks. The tool LogAnalyzer does not really support structured log file analysis, even though some additional fields can be added. This is not a real structured solution and will not be considered in this part of the thesis.

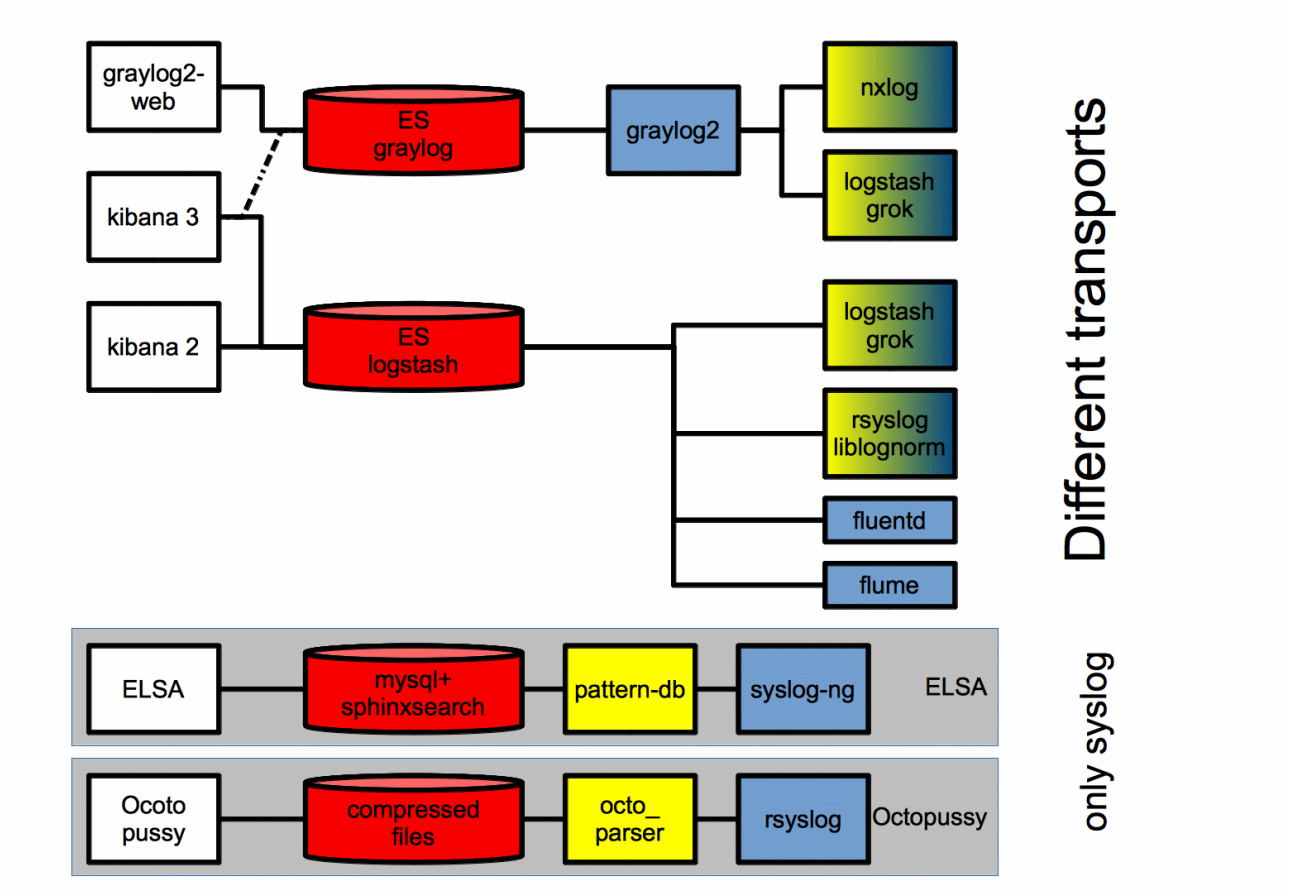

Illustration 12: Possible toolchains (red=storage, yellow=normalizer, white=webpages, blue=shipper

As seen in Illustration 12, ELSA and Octopussy are two special cases, because they are both not modular like the other tools, but they are combining different tools in a predefined way. This makes it harder to build things on top of the tools.

The tools fluentd and flume could be used to write the log data directly to elasticsearch, but then no normalization would be possible. Both are therefore not included in the list of toolchains. Kibana 3 can access elasticsearch in graylog2 format, but only by creating the dashboard completely from scratch, therefore the dashed line.

The webpages can access the data storage in parallel, as they do not make changes to the data. Because of that they will be ignored in the collection of toolchains.

This results in the following toolchains:

- Elasticsearch (graylog format) - graylog2 - logstash
- Elasticsearch (graylog format) - graylog2 - nxlog
- Elasticsearch (logstash format) - logstash (grok)
- Elasticsearch (logstash format) - rsyslog (liblognorm)
- ELSA
- Octopussy

These six toolchains will be discussed in this chapter.

5.2 Toolchain Features

To get a better overview on the different toolchains, some features will be compared. This includes features like high availability, size of rule base and ease of installation.

5.2.1 Accepting structured log files

To select a solution it is necessary to know what kind of log files are going to be processed. This analysis should not only be done for the current log files, but should also include thoughts about what kind of log files need to be processed in the future. If programs start to write structured log files in the future, it would be bad if these messages cannot be used by the selected toolkit.

If the selected solution needs to accept already structured log files, both ELSA and Octopussy can be removed from the selection because they cannot accept already structured log files. The only input into the system is semi-structured syslog messages. Both have the possibility to normalize different kinds of syslog messages, but structured log messages cannot be send in. The idea of de-normalization and re-normlization will be ignored here, because of possible parsing errors and performance reasons.

Program	accepts stuctured log files
Elasticsearch (graylog format) - graylog2 - logstash	yes
Elasticsearch (graylog format) - graylog2 - nxlog	yes
Elasticsearch (logstash format) - logstash (grok)	yes
Elasticsearch (logstash format) - rsyslog (liblognorm)	yes
ELSA	no
Octopussy	no

Table 6: Feature: accepting structured log files

5.2.2 Reliable transport

The log files should not be lost on the way to the central log server. This can be avoided by storing the log data locally should the log server be unavailable.

Both graylog2 and logstash support AMQP for reliable message transfer. Logstash supports a huge number of other reliable inputs. ELSA is based on syslog-ng and only accepts syslog messages. Syslog-ng does not support reliable syslog transfer. Octopussy is using rsyslog to receive log messages, because of the RELP transport of rsyslog it is possible to get reliable syslog transport with Octopussy.

Program	Reliable transport
Elasticsearch (graylog format) - graylog2 - logstash	yes
Elasticsearch (graylog format) - graylog2 - nxlog	yes
Elasticsearch (logstash format) - logstash (grok)	yes
Elasticsearch (logstash format) - rsyslog (liblognorm)	yes
ELSA	no
Octopussy	yes

Table 7: Feature: reliable transport

5.2.3 High availability

Log servers are an important part of the server infrastructure. To make sure the server is always running, it is advisable to create a high availability setup. This is normally done in the form of multiple servers running in parallel and writing to the storage in parallel. In case of a disaster it is helpful to distribute the data to different servers.

Cluster software (like Red Hat, Veritas, VMWare Vmotion) is not considered here, as this is a general solution that can be used with every software.

With graylog2 multiple graylog2 nodes can be created, which access the same elasticsearch cluster. Only one of the graylog2 servers needs to be configured as master, because it has to run some cleanup jobs, but that function can be easily moved. Logstash also supports writing to the same elasticsearch cluster as explained in chapter seven of the logstash book by [Turnbull2013]. ELSA also supports multiple nodes, but these nodes are always writing into their own local database. If a node goes down, it is possible to write to a different node and the data is stored there. A query will be sent to all nodes and therefore all log messages from before and after the problem are integrated into one view. Octopussy does not support a failover solution.

Program	high availability	distributed store
Elasticsearch (graylog format) - graylog2 - logstash	yes	yes
Elasticsearch (graylog format) - graylog2 - nxlog	yes	yes
Elasticsearch (logstash format) - logstash (grok)	yes	yes
Elasticsearch (logstash format) - rsyslog (liblognorm)	yes	yes
ELSA	yes	no
Octopussy	no	no

Table 8: Feature: high availability

5.2.4 User separation and LDAP

A central log system can be very helpful with the correlated and centralized access to all log files, but sooner or later other people outside of the operators will want access to some of the log files. Having some kind of user separation is helpful here, making it possible to give certain users access to only certain kinds of log messages. Defining which user gets access to which log files should be an easy task. To handle user creation and central password management the solution should support LDAP or Active Directory.

Graylog2 makes it possible to create log streams and give users the permission to access the streams as readers. Logstash does not have users and therefore cannot give a limited view of the log messages. ELSA has the possibility to activate user logons. In the default setting everybody that can access the webpage can read all log files. With dashboards it is possible to store queries and give users access to the log messages that this query returns. In Octopussy the user management is very detailed. It is possible to create read-only users that can read all log files, but cannot make any changes to the configuration. There also are restricted users that can be limited to the log data of certain devices, services, alerts or special reports.

Program	User separation	LDAP
Elasticsearch (graylog format) - graylog2 - logstash	yes	yes
Elasticsearch (graylog format) - graylog2 - nxlog	yes	yes
Elasticsearch (logstash format) - logstash (grok)	no	no
Elasticsearch (logstash format) - rsyslog (liblognorm)	no	no
ELSA	yes	yes
Octopussy	yes	yes

Table 9: Feature: User separation and LDAP

5.2.5 Size of rule base

The creation of the rules for log normalization can be a very tedious job. It is nice to be able to access a huge amount of them without writing oneself.

Program	# of prepared rules
Elasticsearch (graylog format) - graylog2 - logstash	0
Elasticsearch (graylog format) - graylog2 - nxlog	4
Elasticsearch (logstash format) - logstash (grok)	0
Elasticsearch (logstash format) - rsyslog (liblognorm)	0
ELSA	~130
Octopussy	~1500

Table 10: Feature: size of rule base

5.2.6 Log Analysis

The centralized logs should be analyzed to detect attacks and other anomalies. But not all tools have this required feature available.

Program	log analysis
Elasticsearch (graylog format) - graylog2 - logstash	limited
Elasticsearch (graylog format) - graylog2 - nxlog	yes
Elasticsearch (logstash format) - logstash (grok)	limited
Elasticsearch (logstash format) - rsyslog (liblognorm)	limited
ELSA	no
Octopussy	yes

Table 11: Feature: log analysis

5.2.7 Install

Even when the installation is only done once, problems during installation are discouraging the use of the program. The installation should run on different Linux distributions, for this thesis three distributions are selected. The toolchains are installed on Red Hat Enterprise Linux (RHEL) 6.4, Debian 7 and Ubuntu 12.04 to test the installation.

Elasticsearch, graylog2 and logstash are all Java based tools. Thus the installation is very easy, as it only needs a Java runtime environment and some basic startup scripts. The difficulty of the installation of graylog2 and logstash is the dependency on a specific version of elasticsearch. Rsyslog is prepackaged for all 3 distributions and is also available in the current version directly from the author. Nxlog is not prepackaged by any distribution, but prepared packages for all three distributions are available from the project website. ELSA is using an installation script. The script did not work during the test for this thesis with Debian 7 and RHEL 6.4, even though they are listed as supported platforms on the webpage [ELSAQuickstart]. Ocotpussy offers a tar.gz archive and a built Debian package. The Debian package works with Debian, as well as with Ubuntu. The tar.gz archive was installable with RHEL6.4. All three installations are described on the [OctopussyInstallation] web page.

Program	Debian 7	Ubuntu 12.04	RHEL 6.4
Elasticsearch (graylog format) - graylog2 - logstash	yes	yes	yes
Elasticsearch (graylog format) - graylog2 - nxlog	yes	yes	yes
Elasticsearch (logstash format) - logstash (grok)	yes	yes	yes
Elasticsearch (logstash format) - rsyslog (liblognorm)	yes	yes	yes
ELSA	no	yes	no
Octopussy	yes	yes	yes

Table 12: Feature: easy to install

5.2.8 Speed

The performance of the log system can have a very large impact on the decision for a log system. Measuring the performance can be very complicated, because of the different architectures of the log toolchains.

For example ELSA is using a batch job based system that accepts messages and stores them in a queue; this queue will be imported once a minute as written in chapter Capabilities on the webpage [ELSA-UserGuide]. The indexing is done "every few hours" for performance reasons. This cannot be fairly tested against an elasticsearch storage, which indexes the data during arrival.

Also the way to distribute load is very different between the toolchains. Where Octopussy can only run on one machine, ELSA can run on many machines, each with its local database. The elasticsearch based tools can distribute the data to many data nodes and share the data between data centers.

The master thesis of [Churilin2013] tries to handle this kind of problem. In this thesis the performance was tested with four different toolchains: Graylog2, logstash, rsyslog are writing to elasticsearch and ELSA is writing to MySQL. The result of the performance test is that ELSA is almost ten times faster than graylog or logstash and still five times faster than rsyslog. Octopussy was not covered in this thesis.

The use of a virtual machine on top of a Windows workstation could lead to a lot of noise in the data. Because no raw data was published it is not possible to check the standard deviation of the tests. The used setup suggests that the standard deviation was quite high. The very limited amount of RAM (only 2GB) per machine, could lead to a big disadvantage against the Java based systems with its large memory requirements.

A fair test of all toolchains would also need to include the normalization phase. To create a fair test it has to be made sure that the number of rules are the same everywhere, otherwise the large pattern size of syslog-ng would be constituted as a disadvantage.

One possibility to handle all the problems would be to test every tool separately, as far as this is possible. Tools that can perform multiple tasks, as logstash, would be tested separately for every task. In case of logstash this would constitute a test for: using it as shipper, writing data to elasticsearch and using it for normalization with grok. With the help of the Force Flow Law it would be possible to calculate the speed of the whole system, but the number of tests to be done for such an endeavor would be too large for this thesis.

Because of all these problems no performance data is published in this thesis.

5.3 Summary

The six possible toolchains all have their advantages and disadvantages. Table 13 is a summary of all the features compared in this chapter. This overview will not be used to declare a winner, because what is important or not always depends on the use case of the company. All six toolchains were running in a test environment for several weeks, without any major incidents. The functionalities differ much between the tools, but all are stable and they can be used on a daily basis.

Program	Reliable transport	high availability	distributed store	User separation	LDAP	# of prepared rules	log analysis	Debian 7	Ubuntu 12.04	RHEL 6.4
Elasticsearch (graylog format) - graylog2 - logstash	yes	yes	yes	yes	yes	0	limited	yes	yes	yes
Elasticsearch (graylog format) - graylog2 - nxlog	yes	yes	yes	yes	yes	4	yes	yes	yes	yes
Elasticsearch (logstash format) - logstash (grok)	yes	yes	yes	no	no	0	limited	yes	yes	yes
Elasticsearch (logstash format) - rsyslog (liblognorm)	yes	yes	yes	no	no	0	limited	yes	yes	yes
ELSA	no	yes	no	yes	yes	~130	no	no	yes	no
Octopussy	yes	no	no	yes	yes	~1500	yes	yes	yes	yes

Table 13: Feature: overview

6 Conclusion

It is already possible to create a centralized structured log file analysis infrastructure and there are a lot of different tools available that can help create such a log infrastructure.

Which tools or toolchains are selected is up to the user to decide. This thesis will only show the advantages and disadvantage of the different tools and the functionalities that they offer. Defining a winner for every use case would be unprofessional.

6.1 Short summary about every major tool

As a short overview here are all major tools with a very short summary about the advantages and disadvantage as experienced by the author:

Syslog-ng: The huge normalization ruleset is very nice, the missing reliable syslog transport because of OpenCore is not.

Rsyslog: Very flexible syslog server, the usability of the normalization tool is limited by the missing rules.

Graylog2: It wants structure, but it will work without it. Can handle permissions of log messages.

Logstash: Swiss army knife of log management. "Everybody sees everything" makes it unusable for some companies.

Node-logstash: Can be a nice Java free alternative for logstash, but it is not there yet.

ELSA: Structured log files cannot be used. The fixation on google and google tools is disconcerting for some users. Designed for great speed.

Octopussy: Everything will be analyzed as a structured log file, but then stored in a normal file which slows searches.

Nxlog: It transports logs and can do a lot of changes and analyzing on the way. The OpenCore nature is not as bad as others.

Heka: Very young, but already very usable, could become a great transporter.

kibana 2: Nice webpage, but will be overshadowed by its newer cousin.

kibana 3: Great flexible web front end for elasticsearch, as soon as it is finished.

statsd + graphite: Often used grapher tool chain, not really covered in this thesis.

mysql: Everybody knows it, MySQL will be replaced by MariaDB.

Elasticsearch: A very easy tool, that eases a lot of problems. The split brain situation can be very dangerous in larger setups, if not handled correctly.

fluentd: Very universal transporter written in Ruby.

flume: Very universal transport written in Java, destined for Hadoop.

6.2 Future

The future for structured log files is very uncertain. The CEE standardization process is dead and no revival is in sight, because of uncertain funding. Project Lumberjack could become the new standard, if the creators push it hard enough. Systemd's Journal could also be the new standard, but

will only be available for Linux. GELF and Logstash are already in use, but are both not real standards. JSON appears to be the only unifying base format everybody is working with, but no decision which fields should be used and how they should be named has been reached yet.

6.3 Optimal toolchain

If the author should dream up a perfect log solution it would be this:

- A webpage for the creation of missing normalization rules like octopussy.
- A prepared normalization ruleset like in pattern-db or octopussy.
- Logstash format for elasticsearch with graylog2 functionalities.
- Both kibana 3 and graylog2 as web front ends possible.
- Easy correlation and analysis based on the normalized data.
- Statsd+graphite graphs are prepared and automatically filled with data.
- Logstash input/output capabilities without the memory overhead of Java.
- A universal structured log format that is used by developers and tools alike.

Appendix:

Abbreviations

Abbreviation	
AMQP	Advanced Message Queuing Protocol
ELSA	Enterprise log search and archive
FIFO	First in - first out
GELF	Graylog Extended Log Format
GIT	Open Source code management tool
GNU	Gnu is Not Unix
GPL	Gnu Public License
GUI	Graphical User Interface
JSON	JavaScript Object Notation
KV	Key Value
LDAP	Lightweight Directory Access Protocoll
RFC	Request for Comment
RHEL	Red Hat Enterprise Linux
SVN	Subversion - an Open Source code management tool
STDIN	Standard Input
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Universal Resource Locator
XML	Extensible Markup Language

Overview

Programm	flat file	directory structure / multiple files with *	STDIN/STDOUT	unix domain socket	named pipe	local Windows eventlog	systemd journal	spool message during downtime	BSD syslog udp (RFC3164)	IETF syslog udp (RFC5424)	IETF syslog tcp (RFC5424)	IETF syslog tcp tls (RFC5425)	TLS encrypted channel	RELP	http	websocket	redis	amqp (QPID,ActiveMQ,RabbitMQ)	Stomp (ActiveMQ,RabbitMQ)	0MQ	gelf	lumberjack
logstash	rw	rw	rw	rw	rw	r			rw	rw	rw		r	r	w	rw	rw	rw	rw	rw	rw	rw
graylog2									r	r	r	r			r			rw			r	
rsyslog	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw			w			w		
syslog-ng	rw		rw	rw	rw				rw	rw	rw	rw	rw					w				
node-logstash	rw		rw						r						w		rw			rw	w	
octopussy									r	r	r	r	r	r								
nxlog	rw		rw	rw		r			rw	rw	rw	rw	rw								w	
Heka	rw	r	rw						r	r	r				r			rw				
woodchuck	r	r	w														w					
awesant	r	r	w	rw													w	w	w			
beaver	r	r	w	w													w	w	w	w		
lumberjack	r	r	r																			w
syslog-shipper	r	r							w				w									
remote_syslog	r	r							w	w			w									
fluentd	rw		rw						r						rw	w	w	rw		w		
flume	rw	r	r	r					r	r	r				r			r				
ncode/logix									r									w				
systemd/ journal2gelf							r														w	
eventlog-to-syslog						r			w													

Table 14: Total Overview: Part 1

Programm	SNMP	statsD	graphite	graphtastic	varnishlog	kafka	mongodb	hadoop	elasticsearch	SQL (DBI or native)	other
logstash	r	w	rw	w	r		w		rw(l)		log4j, irc, twitter, xmpp, email, nagios, amazon
graylog2									rw(g)		
rsyslog	w						w	w	w(l)	N/DBI	
syslog-ng							w			DBI	linux accounting log
node-logstash		w							w(l)		
nxlog										DBI	
Heka			w						w(l)		nagios,
fluentd	r					w	w	w	w(l)	DBI	fnordmetric, couchdb, own log libs
flume								w	w(l)		avro, JMS, HBase, solr, JDBC, irc

Table 15: Total Overview: Part 2