Comparative Analysis of Open-Source Log Management Solutions for Security Monitoring and Network Forensics

Risto Vaarandi, Paweł Niziński
NATO Cooperative Cyber Defence Centre of Excellence, Tallinn, Estonia
risto.vaarandi@ccdcoe.org
pawel.nizinski@ccdcoe.org

Abstract: Nowadays, centralised event log management plays a crucial role in security monitoring and network forensics. While commercial log management solutions are regularly reviewed and compared by independent organisations (e.g. Gartner Magic Quadrant reports), such comparisons are often hard to find for open-source tools, especially for recently created solutions. However, many institutions are using open-source tools for monitoring and forensics, since they allow for implementation of incident detection and analysis frameworks in a cost-efficient way. Furthermore, recently appeared open-source solutions have started a new architectural trend, where the log management system consists of independent and replaceable modules which interact through well-defined interfaces and protocols. In contrast, most commercial systems are essentially monolithic solutions where individual components (such as the GUI or event collector service) cannot be changed for a component from another vendor. In this paper, we will focus on widely used open-source solutions for log management and discuss their recent developments. We will also cover novel technologies and tools which have appeared during the last 2-3 years.

Keywords: log management, network forensics, security monitoring

1. Introduction

Centralised event log management plays a crucial role in security monitoring and network forensics, since it allows for gathering events from thousands of nodes to a few dedicated servers where central analysis is carried out. The analysis can be a real-time process, where security incidents are detected from incoming events through event correlation and other advanced monitoring techniques; it can also be an off-line forensics activity, where past events are investigated in order to study security incidents that have already occurred.

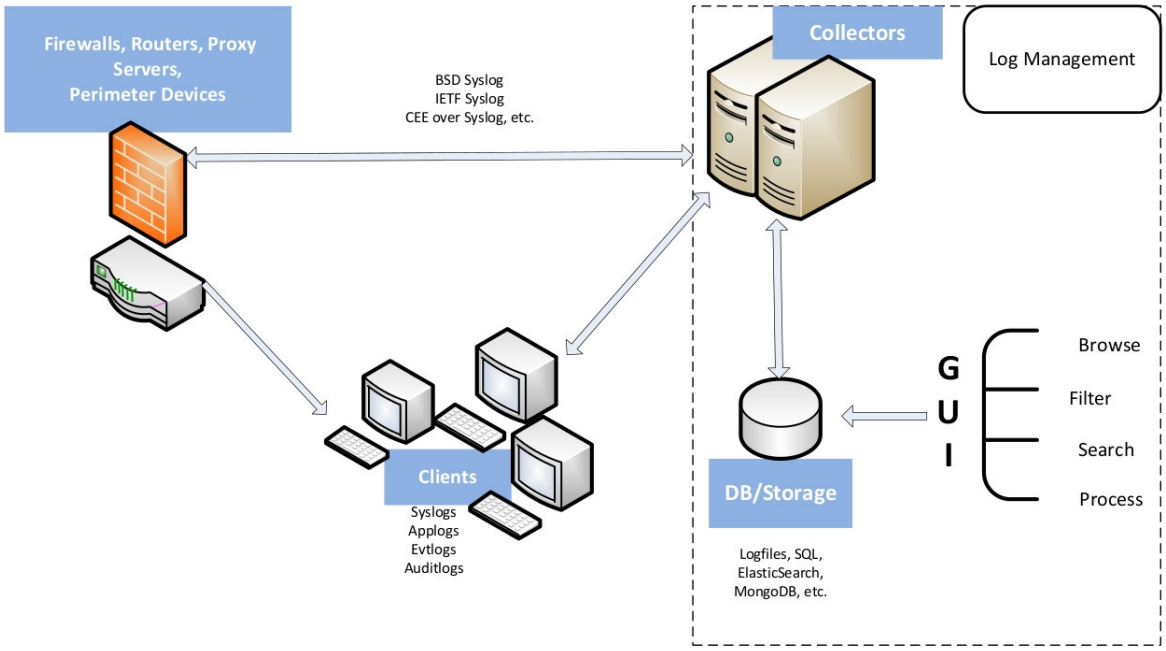

Figure 1: An architectural overview of an event log management framework.

Without event log collection to central location(s), monitoring and forensics activities would have to be carried out

at individual network nodes, which is time-consuming and prevents the timely resolution of security incidents. Furthermore, the attacker may erase events from the local log in order to remove any traces of his/her malicious activities. For the above reasons, a number of commercial and open-source solutions have been created for log collection and centralised analysis. Figure 1 provides an overview of the essential components of an event log management framework.

As depicted in Figure 1, nodes of the IT system are using protocols like IETF syslog for sending events to the collector service(s) at the central log server(s). Collector services use various techniques for filtering and normalising events and store preprocessed events into some means of storage (e.g. a database or a flat file). Human analysts can access stored data through a GUI for carrying out searches, creating reports, and other analytical tasks.

In this paper, we will focus on most prominent open-source log management solutions. The first contribution of this paper is an analytical comparison of presented tools; the second contribution is a detailed comparative performance evaluation of the tools. For this purpose, we conducted a set of experiments for assessing their resource consumption and event processing speed under a heavy load (all experiments were carried out in December 2012). To the best of our knowledge, such performance evaluations have not been conducted recently for state-of-the-art open-source log management solutions.

The remainder of this paper is organised as follows: section 2 provides an overview of log collection protocols and log storing techniques, section 3 provides a discussion and performance evaluation for leading open-source syslog servers, section 4 presents an overview and performance assessment of graphical log management systems, and section 5 concludes the paper.

2. Log collection and storing

Until the 1980s, event logging was mostly accomplished by writing events to a file on a local disk. In the 1990s, UDP-based BSD syslog protocol became widely used for log collection (Lonvick 2001). The protocol defines 24 message facility (sender type) and 8 severity values which range from 0 to 23 and 0 to 7 respectively. For reasons of convenience, textual acronyms are often used instead of numerals, e.g. *daemon* denotes facility 3 and *warning* denotes severity 4. According to BSD syslog, the payload of the UDP packet carrying the message must have the format *<Priority>Timestamp Hostname MSG*, where *Priority* is defined as a numeral *8*facility_value + severity_value*. For example, the following message represents a warning “ids[1299]: port scan from 192.168.1.102” for *daemon* facility which was issued on November 17 at 12:33:59 by the network node myhost2:

<28>Nov 17 12:33:59 myhost2 ids[1299]: port scan from 192.168.1.102

By convention, the alphanumerals that start the *MSG* field are regarded as the *Tag* subfield which represents the name of the sending program (“ids” in the above example), while the remainder of the *MSG* field is regarded as the *Content* subfield (“[1299]: port scan from 192.168.1.102” in the above example).

Despite its popularity, BSD syslog protocol has a number of drawbacks which are summarised below:
1) no support for reliable message transmission over TCP;
2) no support for encryption and authentication;
3) timestamps are not specific enough, lacking the timezone, year, and fractions of a second;
4) apart from *Tag* and *Content* subfields, the *MSG* field has no structure.

In order to make message transmission more reliable, a TCP flavor of BSD syslog protocol was proposed during the previous decade, where a stream of newline-separated messages in BSD syslog format is sent over a TCP connection. In 2009, more advanced IETF syslog protocol was introduced that addresses all drawbacks of BSD syslog (Gerhards 2009; Miao, Ya and Salowey 2009; Okmianski 2009). The IETF syslog supports secure message transmission over TLS, and uses a new message format with more detailed RFC3339 timestamps (Klyne and Newman 2002) and structured data blocks. The following example depicts the previous sample message in a new format:

<28>1 2012-11-17T12:33:59.223+02:00 myhost2 ids 1299 - [timeQuality tzKnown="1" isSynced="1"] port scan from 192.168.1.102

The priority specification *<28>* is followed by the protocol version number (1). In addition, the sender is passing a structured data block *timeQuality* to the receiver, indicating that sender's clock is synchronised to an external reliable time source.

Another currently ongoing effort to introduce structure to log messages is the Common Event Expression (CEE) initiative. CEE has proposed JSON and XML formats for events, while also suggesting the use of BSD and IETF syslog protocols for transporting JSON-formatted events (CEE 2012). In addition, there are some application-specific protocols for structured logging, e.g. GELF.

When logs are collected to central server(s), they need to be written to a permanent storage. In many cases, incoming log messages are written into flat files on the disk of the central server. While this consumes little CPU time and allows for receiving large volumes of events per second, searching relevant events from flat files can be time consuming and resource intensive. Therefore, log messages are often stored into SQL databases that facilitate fast and efficient searching. However, each database table contains a fixed number of columns, with each column representing a certain message field of a fixed type (e.g. integer or string). As a consequence, the fields of a log message have to comply with the structure of a database table that is holding the log data. In order to address this requirement, fields of all collected messages must be well known in advance, so that appropriate database schema can be defined.

Unfortunately, if logs are received from a wide variety of sources, log messages in previously unseen formats are likely to appear. In order to address this problem, *document-oriented databases* have emerged recently as alternative log storage solutions. Although the implementations of document-oriented databases vary, they can be viewed as a collection of *documents*, where each document is usually a record of fieldname-value pairs. It is important to note that each inserted document can have a unique set of fields which do not need to be known in advance.

During the last 1-2 years, Java-based *elasticsearch* has emerged as one of the most widely used document-oriented database engines for storing log data (Elasticsearch 2013). *Elasticsearch* accepts new documents in JSON format over a simple HTTP interface, inserting the incoming document into a given index and thus making the document searchable for future queries. Support for distribution is built into the core of *elasticsearch* and several instances of *elasticsearch* engines can be easily joined into a single cluster. Furthermore, each database index can be split into so-called *shards,* which can be located at different cluster members. Also, each index can have one or more *replicas* for implementing a fault tolerant cluster.

3. Syslog servers

In this section, we will cover leading open-source syslog servers *rsyslog* (Rsyslog 2013), *syslog-ng* (Syslog-ng 2013) and *nxlog* (Nxlog 2013). The discussion and experiments presented in this section are based on *rsyslog* 7.2.3, *syslog-ng* 3.3.7 and *nxlog* ce-2.0.927.

3.1 Rsyslog, syslog-ng and nxlog

Rsyslog, *syslog-ng* and *nxlog* have been designed to overcome the weaknesses of traditional UNIX *syslogd* server which supports only BSD syslog protocol, and is able to match and process messages by facility and severity. *Rsyslog*, *syslog-ng* and *nxlog* support not only such simple message matching, but advanced message recognition with regular expressions, conversion of messages from one format to another, authenticated and encrypted communications over IETF syslog protocol, etc. *Syslog-ng* and *nxlog* have also a commercial edition with extended functionality. *Rsyslog* and *syslog-ng* run on UNIX platforms, while *nxlog* is also able to work on Windows.

The configuration of all servers is stored in one or more textual configuration files. *Syslog-ng* uses a highly flexible and readable configuration language which is not compatible with UNIX *syslogd*. The message sources, matching conditions and destinations are defined with named blocks, with each definition being reusable in other parts of the configuration. *Syslog-ng* is also well documented, featuring a detailed administrator's manual consisting of hundreds of pages. This makes it easy to create fairly complex configurations, even for inexperienced users.

Rsyslog is an efficient syslog server that was specifically designed for handling heavy message loads (Gerhards 2010). *Rsyslog* uses a quite different configuration language that supports UNIX *syslogd* constructs. This allows for easy migration of old *syslogd* setups to *rsyslog* platform. Also, there are many additional features in the *rsyslog* configuration language. Unfortunately, over time, several different syntaxes have been included in the language which has introduced inconsistencies (Gerhards 2012). Functionality-wise, *rsyslog* supports several highly useful features not present in open-source versions of *syslog-ng* and *nxlog*. Firstly, it is possible to set up temporary message buffering to local disk for log messages which were not sent successfully over the network. The buffering is activated when the connection with a remote peer is disrupted, and when the peer becomes available again, all buffered messages are retransmitted. Secondly, the latest stable release of *rsyslog* has an efficient support for *elasticsearch* database (Rsyslog-ver7 2012).

Nxlog uses the Apache style configuration language. As with *syslog-ng*, message sources, destinations and other entities are defined with named blocks which allows them to be reused easily. Also, *nxlog* has a solid user manual. The advantages of *nxlog* over other syslog servers include native support for Windows platform and Windows Eventlog. Also, *nxlog* is able to accept input events from various sources not directly supported by other servers, including SQL databases and text files in custom formats. Finally, *nxlog* is able to produce output messages in the GELF format, allowing for seamless integration with the *Graylog2* log visualisation solution.

In order to illustrate the differences between the configuration languages of syslog-ng, rsyslog and nxlog, we have provided configuration statements in three languages for the same log processing scenario:

```
##### configuration for syslog-ng

@version:3.3

source netmsg { udp(port(514)); };
filter ntpmsg { program('^ntp') and level(warning..emerg); };
destination ntplog { file("/var/log/ntp-faults.log"); };

log { source(netmsg); filter(ntpmsg); destination(ntplog); };

##### configuration for rsyslog

$ModLoad imudp
$UDPServerRun 514

if re_match($programname, '^ntp') and $syslogseverity <= 4 then {
  action(type="omfile" file="/var/log/ntp-faults.log")
}

##### configuration for nxlog

<Input netmsg>
   Module  im_udp
   Host       0.0.0.0
   Port       514
   Exec       parse_syslog_bsd();
</Input>

<Output ntplog>
   Module   om_file
   File        "/var/log/ntp-faults.log"
   Exec       if $SourceName !~ /^ntp/ or $SyslogSeverityValue > 4 drop();
</Output>

<Route ntpfaults>
   Path       netmsg => ntplog
</Route>
```

First, the above configurations tell syslog servers to accept BSD syslog messages from UDP port 514 (in the case of *syslog-ng* and *nxlog*, the name *netmsg* is assigned to this message source). Then, the message filtering condition is defined for detecting messages with the *Tag* field matching the regular expression *^ntp* (in other words, the name of the sending program begins with the string "ntp"), and with the message severity falling between *warning* (code 4) and *emerg* (code 0). Note that for *nxlog*, the inverse filter is actually used for dropping irrelevant messages. Finally, the file /var/log/ntp-faults.log is used as a destination for storing messages that have passed the filter (in the case of *syslog-ng* and *nxlog*, the name *ntplog* is assigned to this destination).

3.2 Experiments for evaluating the performance of rsyslog, syslog-ng and nxlog

In order to evaluate how well each server is suited for the role of a central syslog server, we conducted a number of experiments for assessing their performance. During the experiments we used three benchmarks for stress-testing the servers, and measured the CPU time consumption and overall execution time of each server during every test run. We call the benchmarks BSD-Throughput, IETF-Throughput and Filter-Throughput, and define them as follows:

1) BSD-Throughput – 1 client sends 10,000,000 plain-text BSD syslog messages to the syslog server over TCP. The messages are written to one log file without filtering.
2) IETF-Throughput – 1 client sends 10,000,000 encrypted IETF syslog messages to the syslog server over TCP. The messages are written to one log file without filtering.
3) Filter-Throughput – there are 5 clients, each sending 2,000,000 plain-text BSD syslog messages to the syslog server over TCP. The messages are identical to messages of the BSD-Throughput benchmark which allows for making performance comparisons between two benchmarks. The server is configured to process incoming log data with 5 filters and to write messages into 5 log files. All filters include regular expression match conditions for the message text (*Content* field) and/or program name (*Tag* field), and some filters also have additional match conditions for message facility and severity.

Table 1: Comparative performance of rsyslog, syslog-ng and nxlog.

	rsyslog	syslog-ng	nxlog
BSD-Throughput maximum, minimum and average CPU time consumption (seconds)	17.994 14.601 16.024	97.094 88.942 94.090	86.809 83.929 85.100
BSD-Throughput maximum, minimum and average execution time (seconds)	12.778 10.736 11.853	98.961 90.715 96.079	54.261 52.253 53.281
IETF-Throughput maximum, minimum and average CPU time consumption (seconds)	47.190 41.448 43.883	115.684 106.813 111.823	166.455 161.536 164.605
IETF-Throughput maximum, minimum and average execution time (seconds)	33.268 30.184 31.337	128.055 108.911 115.084	71.605 69.434 70.404
Filter-Throughput maximum, minimum and average CPU time consumption (seconds)	50.265 45.626 47.661	216.093 211.143 213.683	2237.954 2191.502 2216.758
Filter-Throughput maximum, minimum and average execution time (seconds)	44.389 39.941 41.624	60.496 58.886 59.792	715.320 701.210 707.933

Note that *rsyslog* and *nxlog* always run in multi-threading mode, while for *syslog-ng* this mode has to be enabled manually. During the testing we discovered that for BSD-Throughput and IETF-Throughput *syslog-ng* performance decreased in multi-threading mode (according to the *syslog-ng* manual, this mode yields performance benefits in the presence of many clients, filters and message destinations). Therefore, we ran *syslog-ng* in a default single-threaded mode for BSD-Throughput and IETF-Throughput tests. Also, we discovered that the tested *nxlog* version was not able to handle IETF syslog messages as required by RFC5425 – in a stream of incoming messages, only the first syslog frame was properly recognised. Also, the tested version was not able to parse some timezone specifications in RFC3339 timestamps. For these reasons, we modified the IETF-Throughput benchmark for *nxlog*, so that instead of proper RFC5425 frames, a stream of newline-separated IETF messages was sent to *nxlog* over TLS connection (this unofficial data transmission mode is supported by all tested servers as an extension to standard modes). The tests were carried out on a Fedora Linux node with 8GB of memory and an Intel Core i5 650 processor. We repeated each test 10 times, and the results are presented in Table 1.

The results reveal several interesting aspects of server performances. Firstly, the performance of *rsyslog* is superior to other servers, both in terms of raw message throughput from single client and efficiency of message filtering for multiple clients. Also, *rsyslog* is able to share its workload between several CPU cores with multi-threading, and thus the execution times are less than overall consumed CPU times. Multi-threading is used very efficiently by *nxlog*, resulting in execution times being 1.5-3 times lower than used CPU time. Unfortunately, the performance of *nxlog* filters is poor – compared with the BSD-Throughput test, the average CPU time consumption increased about 26 times. In contrast, CPU time consumption for *syslog-ng* increased only 2.27 times, while the average execution time actually decreased by almost a half due to the manually enabled multi-threading mode.

4. Log visualisation and preprocessing applications

While the use of databases for storing log messages facilitates fast searching with flexible query language, it is tedious and time-consuming for the user to write separate queries for each search, report or other analytical task. Furthermore, the output from database queries is textual and the user would have to use a separate tool, or even programming language, for visualising this output. For solving this problem, several open-source log visualisation applications have been developed during the last 2-3 years, which are all using *elasticsearch* as their main database engine. In this section, we will review and conduct performance evaluation experiments for *logstash* (version 1.1.5), *Graylog2* (version 0.9.6) and *Kibana* (version 0.2.0).

4.1 Logstash

Logstash is a Java-based utility where a graphical user interface and embedded *elasticsearch* engine are encapsulated into a standalone jar-file (Logstash 2013). This eases the installation of *logstash* since the user does not have to download and install all product components separately. One advantage of *logstash* is its support for many different input and output types. Currently, there are input plugins for accepting syslog messages over TCP and UDP, but also for many other messaging protocols like AMPQ, RELP, GELF, IRC, XMPP, twitter, etc. Among outputs, other monitoring and visualisation systems are supported, including *Nagios*, *Zabbix*, *Loggly*, *Graphite* and *Graylog2*.

Another advantage of *logstash* is a number of different message filter types which allow for flexible recognition, filtering, parsing and conversion of messages. For instance, it is possible to convert multi-line messages into single line format, filter out messages with regular expressions, add new fields to messages from external queries and accomplish many other advanced message manipulation tasks.

One of the most commonly used *logstash* filter types is *grok*. While most log management tools use regular expression language for message matching and parsing, *grok* filters employ many predefined patterns that represent regular expressions for common matching tasks (GrokPatterns 2013). For instance, the pattern PROG is defined as the regular expression *(?:[\w._/%-]+)* and is designed to match the name of the logging program. Using predefined *grok* patterns, a person who is not familiar with the regular expression language can accomplish event parsing tasks in an easier way.

In order to use the GUI of *logstash*, it must be configured to insert events into its embedded *elasticsearch*

database. With the GUI it is possible to carry out basic searches from log messages in the embedded database. Unfortunately, compared with other log visualisation tools, the GUI of *logstash* has quite limited functionality. However, since *logstash* has powerful event filtering and conversion capabilities, it is used mostly as an event preprocessor for different systems, including other log visualisation systems.

4.2 Graylog2

Graylog2 (Graylog2 2013) is a log management system which consists of a Java-based server and a web interface written in Ruby-on-Rails. *Graylog2* server can receive BSD syslog messages over UDP and TCP, but it also features its own GELF protocol that facilitates structured logging (GELF 2013). In addition, *Graylog2* can accept syslog and GELF messages over the AMPQ messaging protocol. Unfortunately, *Graylog2* is not able to parse structured IETF syslog messages and recognise already defined fieldname-value pairs. For BSD syslog messages, *Graylog2* can recognise the standard *Priority*, *Timestamp*, *Hostname* and *MSG* fields, but cannot by default parse the unstructured *MSG* field.

The parsing problem for unstructured messages can be overcome in several ways. First, *Graylog2* server supports message parsing and rewriting through Drools Expert rules and regular expressions (Graylog2-Drools 2013). Second, many sites use *logstash* for receiving syslog messages, parsing out relevant message fields with *grok* filters, and finally sending the parsed messages in structured GELF format to *Graylog2*. Finally, since the *nxlog* syslog server supports the GELF protocol, it can be used as a frontend for receiving both encrypted and plain-text syslog messages and converting them into GELF messages.

For storing parsed messages, *Graylog2* uses *elasticsearch* as its main backend. Unfortunately, all log messages are stored into a single index, which might cause performance issues as many log messages are inserted into it over time. This issue can be addressed with creating a separate *elasticsearch* index for each week or day (the latter technique is used by *logstash*). Fortunately, the developers of *Graylog2* are aware of this problem and it is supposed to be fixed in the next version.

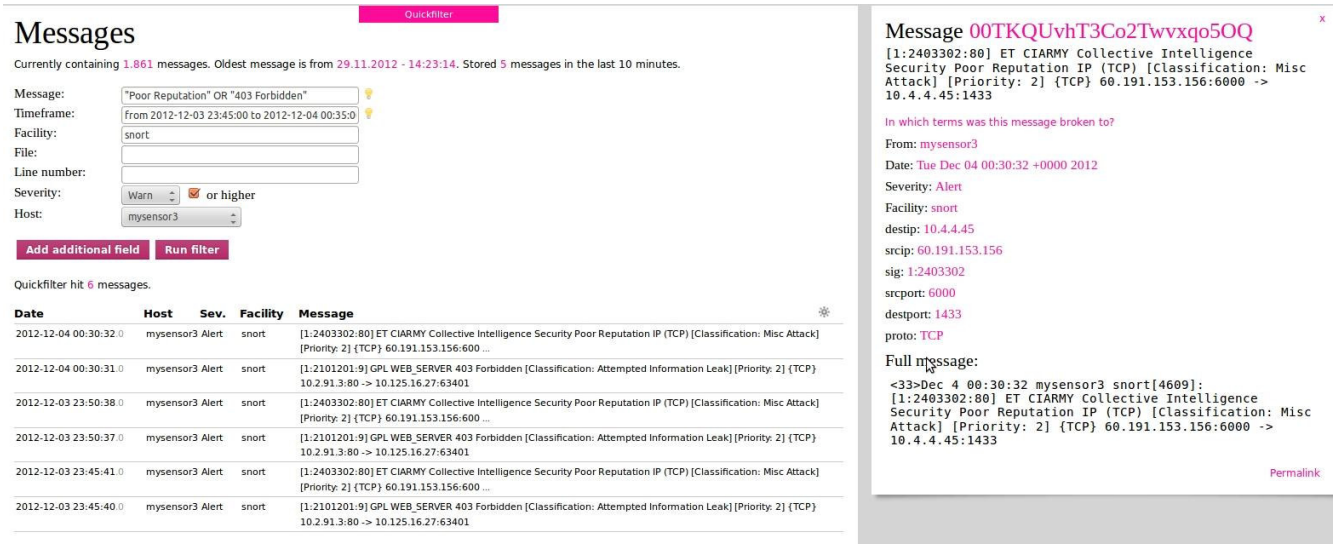

Figure 2: Graphical user interface of Graylog2.

In order to visualise collected log data, *Graylog2* provides a well-written and comprehensive web interface. For accessing the interface, different password-protected user accounts can be set up, with each user having either full or limited rights (the remainder of the discussion will concern the user interface with full admin rights). The interface is divided into several parts. The message view (see Figure 2) allows for getting an overview of stored log messages, presenting the messages in a browser with the most recent messages coming first. In the browser, the timestamp, host, severity, facility and message text fields are displayed. By clicking on an individual message, a detailed view of the message is provided which contains all fieldnames with values (see the right-hand part of Figure 2). Clicking on each individual value will carry out a search for messages with the same

fieldname-value pair, and discovered messages will be displayed in the main message browser. Message search can also be initiated through a separate 'Quickfilter' button in the message view which allows for specifying more than one search condition. For searching a message text field, the Apache Lucene query syntax can be used (Lucene 2012). It supports searches for individual strings, approximate string matching based on Levenshtein distance, proximity searches (e.g. find two strings which have up to 10 words in between), and combining individual search conditions with Boolean operators. Figure 2 depicts an example search for Snort IDS alarms.

Apart from viewing all messages, the user can configure streams that are collections of messages satisfying some message filtering condition. Streams are updated with matching incoming messages in real-time. Messages under each stream can be viewed separately, and also it is possible to configure thresholding and alarming conditions for each stream (e.g. send an alarm to a security administrator if more than 10 messages have appeared under the stream during 1 minute). In order to define a filtering condition for a stream, message field values can be compared with fixed values, but in the case of some fields, also with regular expressions. In addition to streams, *Graylog2* also contains a so called analytics shell which supports a flexible query language for finding individual messages and for creating various reports. Unfortunately, currently all reports are text-based, although in the future releases support for graphical reporting might be added.

During our experiments, we attempted to establish the performance of *Graylog2* in terms of event throughput. We expected the performance of *Graylog2* to be significantly slower than for syslog servers tested in the previous section. First, both *Graylog2* server and *elasticsearch* database engine are written in Java, while *rsyslog*, *syslog-ng* and *nxlog* are coded in C. Second, *Graylog2* server has to insert log messages into *elasticsearch* index, which requires considerably more CPU time than writing messages into flat files. During the tests, we ran *Graylog2* on a Fedora Linux node with 8GB of memory and an Intel Core i5 650 processor. We set up one client for *Graylog2*, which was issuing large amounts of BSD syslog messages over TCP. The combined performance of *Graylog2* server and *elasticsearch* backend was measured in terms of message transmission throughput observed at the client side. During several tests, we were able to reach a throughput of 3,500 messages per second. This illustrates that one *Graylog2* server instance is not scalable to very large environments with many thousands of logging hosts and heavy message loads. Fortunately, the developers are planning to add support into *Graylog2* for multiple server instances, which should substantially increase its overall scalability.

4.3 Kibana

Kibana is another application for visualising collected log data (Kibana 2013). Unlike *Graylog2*, *Kibana* consists only of a Ruby-based web interface which uses *elasticsearch* as a backend, and there is no server to receive log messages over the network and store them into a database. For this reason, *Kibana* cannot run as a standalone system, but has to be used with an application that receives, parses, and stores log messages into *elasticsearch*. Many sites are employing *logstash* for this task, and *Kibana*'s default configuration is *logstash*-compliant. Also, *Kibana* expects that log messages in *elasticsearch* database have some fields that are created by *logstash* (for example, *@timestamp* and *@message*). However, if another application is configured to insert log messages into *elasticsearch* database with these fieldname-value pairs, *Kibana* is able to work on stored log data.

In order to search log messages, *Kibana* supports full Apache Lucene query syntax for all message fields (Figure 3 depicts an example search for Snort IDS alarm data). One advantage of *Kibana* over *Graylog2* is the support for the creation of various graphical reports. Reports can be created based on a selected message field and time frame, either for all messages or for some message matching criteria. *Kibana* supports the creation of pie charts which reflect the distribution of field values, trend analysis reports and count reports for field values. By selecting some value from the report form, the user can go to relevant log messages. Reports can also be created directly from searches – for example, the query *@fields.srcip=10.1.1.1* selects all messages where the field @fields.srcip (source IP address) has the value 10.1.1.1, while the query

@fields.srcip=10.1.1.1 | terms @fields.dstip

creates a pie graph about the distribution of @fields.dstip (destination IP address) values for source IP 10.1.1.1.

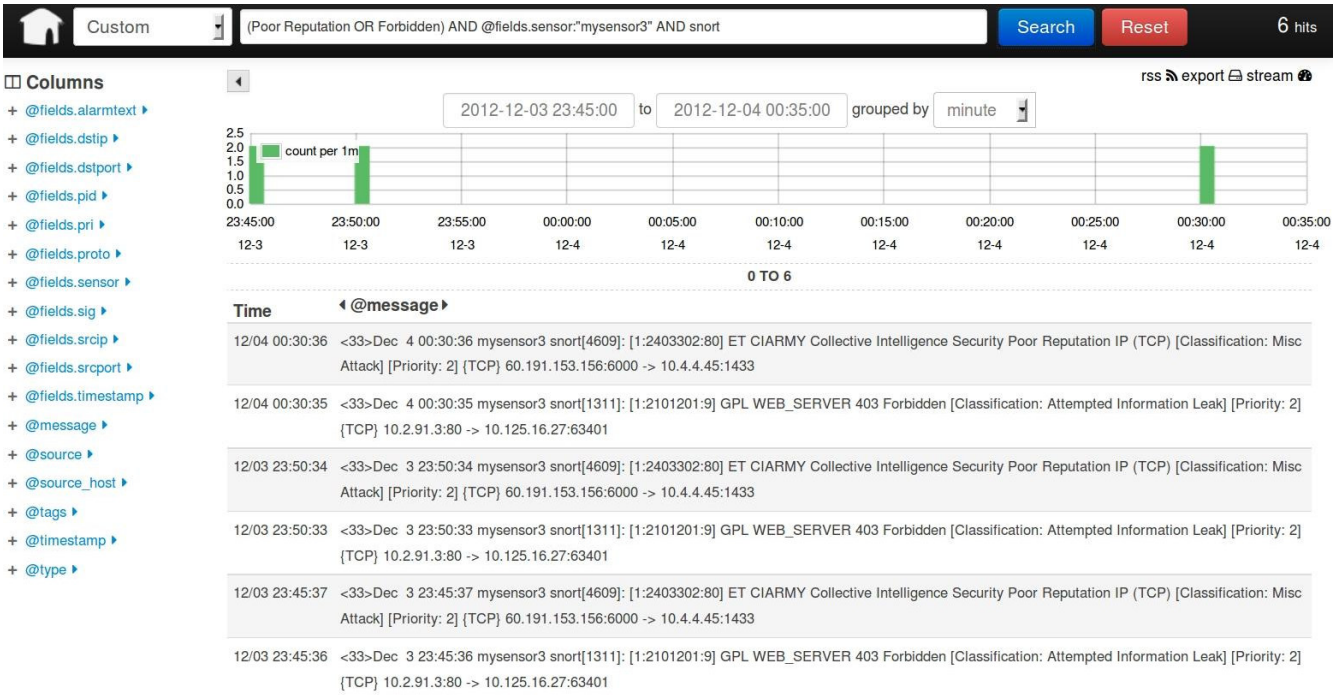

Figure 3: Graphical user interface of Kibana.

Since *rsyslog* has had *elasticsearch* support since 2012, it can be used instead of *logstash* for receiving and preparing log data for *Kibana*. In order to assess the performance of *logstash* and *rsyslog*, we installed *Kibana* with *elasticsearch* on a Fedora Linux node with 8GB of memory and an Intel Core i5 650 processor, and set up both *rsyslog* and *logstash* at this node. Both solutions were configured to insert messages into *elasticsearch* in bulk mode (for *rsyslog*, the message batch size was 16, while for *logstash* a batch size of 100 was used). For performance evaluation, we sent 100,000 BSD syslog messages over TCP to the receiver, and measured the processing time of these messages. At the end of each test, a query was made to *elasticsearch* for verifying that all messages were successfully inserted into the database. We repeated this test 100 times for *rsyslog* and *logstash*, deleting all inserted messages between consecutive test runs. The results of our experiments are presented in Table 2. For *rsyslog*, 100,000 messages were processed in an average of 17.066 seconds, yielding the average processing speed of 5859.6 messages per second. In the case of *logstash*, the average processing speed was 1732.952 messages per second. In other words, *rsyslog* is able to insert messages into *elasticsearch* more than 3 times faster.

Table 2: Comparative performance of rsyslog and logstash for elasticsearch bulk insert operations.

	Minimum processing time (seconds)	Maximum processing time (seconds)	Average processing time (seconds)	Average event processing speed (events per second)
rsyslog	15.998	21.031	17.066	5859.604
logstash	56.084	73.695	57.705	1732.952

5. Summary

In this paper, we have reviewed a number of widely used and efficient open-source solutions for collecting log data from IT systems. We have also described some novel technologies and setups for tackling the logging in large networks. One of the major contributions of this paper is the performance evaluation of described solutions through a series of benchmarks which mimic heavy workload in real-life environments. Through the experiments

conducted, we have been able to identify specific advantages and weaknesses of each tool (e.g. efficient multi-threading or event filtering). Although our tests indicate that some tools have superior performance under specific circumstances, each tool offers a unique set of features to the end user. Therefore, the selection of a log management tool depends heavily on the specific nature of the environment. In order to automate the experiments for evaluating log management tools, we have written a simple toolkit consisting of a few Perl and UNIX shell scripts. For future work, we plan to elaborate our testing tools and release them to the public domain.

References

CEE (2012) *Common Event Expression, version 1.0beta1*, http://cee.mitre.org/language/1.0-beta1/

Elasticsearch (2013) http://www.elasticsearch.org

GELF (2013) http://www.graylog2.org/about/gelf/

Gerhards, R. (2009) *The Syslog Protocol*, RFC5424, http://www.ietf.org/rfc/rfc5424.txt

Gerhards, R. (2010) “Rsyslog: going up from 40K messages per second to 250K”, Linux Kongress 2010, http://www.gerhards.net/download/LinuxKongress2010rsyslog.pdf

Gerhards, R. (2012) *BSD-Style blocks will go away in rsyslog v7*, http://blog.gerhards.net/2012/09/bsd-style-blocks-will-go-away-in.html

Graylog2 (2013) http://graylog2.org

Graylog2-Drools (2013) https://github.com/Graylog2/graylog2-server/wiki/Message-processing-rewriting

GrokPatterns (2013) https://github.com/logstash/logstash/blob/master/patterns/grok-patterns

Kibana (2013) http://kibana.org

Klyne, G. and Newman, C. (2002) *Date and Time on the Internet: Timestamps*, RFC3339, http://www.ietf.org/rfc/rfc3339.txt

Logstash (2013) http://logstash.net

Lonvick, C. (2001) *The BSD syslog Protocol*, RFC3164, http://www.ietf.org/rfc/rfc3164.txt

Lucene (2012) *Apache Lucene – Query Parser Syntax*, https://lucene.apache.org/core/old_versioned_docs/versions/3_5_0/queryparsersyntax.html

Miao F., Ya, M. and Salowey J. (2009) *Transport Layer Security (TLS) Transport Mapping for Syslog*, RFC5425, http://www.ietf.org/rfc/rfc5425.txt

Nxlog (2013) http://http://nxlog-ce.sourceforge.net/

Okmianski, A. (2009) *Transmission of Syslog Messages over UDP*, RFC5426, http://www.ietf.org/rfc/rfc5426.txt

Rsyslog (2013) http://www.rsyslog.com

Rsyslog-ver7 (2012) *Main Advantages of rsyslog v7 vs v5*, http://www.rsyslog.com/main-advantages-of-rsyslog-v7-vs-v5/

Syslog-ng (2013) http://www.balabit.com/network-security/syslog-ng/