TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

LASSI KOJO

SITUATION AWARENESS IN COMPUTER SYSTEMS

Master of Science thesis

Examiner: Prof. Billy Bob Brumley
Examiner: MSc. Markku Vajaranta
Examiners and topic approved by the
Faculty Council of the Faculty of
Information Technology
on 27th September 2017

ABSTRACT

LASSI KOJO: Situation Awareness in Computer Systems
Tampere University of Technology
Master of Science thesis, 48 pages, 9 Appendix pages
November 2017
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Prof. Billy Bob Brumley
Examiner: MSc. Markku Vajaranta
Keywords: Information Security, SIEM, Log Management, Network Monitoring

As the complexity of networks increases, new tools need to be implemented in order to maintain control over the connected devices. The thesis presents a way to reach situation awareness in computer system in a cost-effective way without compromising flexibility and scalability. The definition in situation awareness in cyber security context includes i.e. that one needs to be aware of the current situation, how situations evolve and why and how the current situation is caused. In order to achieve situation awareness, two tools are presented: monitoring system and log analytics platform.

Monitoring system is a proactive system which keeps track of status about all the devices and services configured to be monitored. The status and received events are stored for later usage, and graphs are drawn based on values of different services and statuses. Log analytics platform is a reactive system which provides insight into structured and enriched log data. It can visualize the log data, analyze and alarm based on pre-defined rules and utilize machine learning for anomaly detection.

These two systems are integrated together using alarming feature of the monitoring system, so that logs can be linked to the exact device in monitoring system, hence collecting the relevant data in one centralized view so that the incidents can be investigated further on log analytics platform. Together they provide deep insight into the computer system and enable situation awareness.

TIIVISTELMÄ

LASSI KOJO: Tilannekuvan muodostaminen tietojärjestelmistä
Tampereen teknillinen yliopisto
Diplomityö, 48 sivua, 9 liitesivua
lokakuu 2017
Tietotekniikan koulutusohjelma
Pääaine: Pervasive Systems
Tarkastaja: Prof. Billy Bob Brumley
Tarkastaja: MSc. Markku Vajaranta
Avainsanat: Tietoturvallisuus, SIEM, Lokienhallinta, Verkonvalvonta

Tietoverkkojen monimutkaisuuden kasvaessa paine uusien työkalujen käyttöönottoon kasvaa, jotta ylläpitäjät voisivat jatkossakin säilyttää verkkoon kytkettyjen laitteiden hallinnan omissa käsissään. Tässä diplomityössä esitellään kustannustehokas ja helposti laajennettava järjestelmä tilannekuvan muodostamiseen tietojärjestelmässä. Tilannekuvan määritelmä kyberturvallisuuden kontekstissa sisältää mm. vaatimukset nykyisen tilanteen ymmärtämisestä, siitä miten erilaiset tilanteet kehittyvät, ja miksi ja miten nykyinen tilanne on aiheutunut. Jotta voitaisiin muodostaa tilannekuva tästä hetkestä, esitellään kaksi työkalua: valvontajärjestelmä ja lokianalytiikka-alusta.

Valvontajärjestelmä on proaktiivinen järjestelmä, joka pitää kirjaa kaikkiin siihen liitettyjen laitteiden ja palveluiden tilasta. Kysytty tilatieto ja vastaanotetut viestit eri järjestelmistä tallennetaan myöhempää käyttöä varten, ja kerätystä tiedosta piirretään kuvaajia. Lokianalytiikka-alusta on reaktiivinen järjestelmä, joka tarjoaa matalan tason näkyvyyden jäsenneltyyn ja rikastettuun lokidataan. Se voi visualisoida lokidataa, analysoida ja hälyttää ennaltamääriteltyjen rajojen perusteella, tai hyödyntää koneoppimista poikkeuksien havainnointiin.

Nämä kaksi järjestelmää on integroitu keskenään valvontajärjestelmän hälytysominaisuutta hyödyntäen, joka yhdistää lokianalytiikka-alustasta lähetetyt hälytysviestit valvontajärjestelmässä olevaan laitteeseen tai palveluun. Näin ollen kaikki tarpeellinen tieto on keskitetty yhteen näkymään, jonka hälyttäessä voidaan porautua syvemmälle lokianalytiikka-alustan säilyttämiin lokeihin ongelman ratkaisemiseksi. Yhdessä nämä kaksi järjestelmää tarjoavat läpivalaisevan näkymän tietojärjestelmään ja mahdollistavat tilannekuvan muodostamisen.

PREFACE

I have been working at JMJping Oy since the first year of my studies in TUT. The time spent there has had a huge effect on my career and has guided me to study subjects which I am really interested in, which I had really no clue before the second year of working. This thesis concludes my work at JMJping Oy and the major part that I have been doing for them for the past four years.

I would like to express my deep gratitude especially to my supervisor Janne Tapio for all the opportunities and responsibilities I have received over the years in all sorts of projects, they have made me a better engineer. I also appreciate the support from Marko Toivola, and all the other co-workers I have had during my time there.

During my studies, I had a chance to get to know the staff of TUT Cyber Security Lab, especially MSc. Markku Vajaranta. Markku has helped me a lot not only supervising this thesis but also in other courses and we have had great dialogs about the work of a system administrator.

I also had a chance to work with a great computer security team at CERN supervised by Dr. Stefan Lüders in the summer of 2015. It was an amazing experience not only because of the size of the environment but also because of the people working there.

This thesis is the climax of six and a half years of studying and working at the same time and for now the last academic work I plan to do before I head to the business world full-time.

Finally, I would like to thank Prof. Billy Bob Brumley for his blazing fast handling of the administrative side of this thesis and his valuable feedback.

Tampere, 13.11.2017

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
CA	Certificate Authority
CEF	Common Event Format
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EAP-TLS	Extensible Authentication Protocol-Transport Layer Security
GPO	Group Policy Object
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICS	Industrial Control System
IDS	Intrusion Detection System
IPFIX	IP Flow Information Export
JSON	JavaScript Object Notation
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LQL	Livestatus Query Language
MAC address	Media Access Control address
NAS	Network-attached storage
NAT	Network Address Translation
NPS	Microsoft Network Policy Server
PLC	Programmable Logic Controller
PSU	Power Supply Unit
RADIUS	Remote Authentication Dial In User Service
RAID	Redundant Array of Inexpensive Disks
RDP	Remote Desktop Protocol
SA	Situation Awareness
SCADA	Supervisory Control And Data Acquisition
SIEM	Security Information and Event Management
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UPS	Uninterruptible Power Supply

VPN	Virtual Private Network
YAML	YAML Ain’t Markup Language

1. INTRODUCTION

Information security has become an extremely hot topic in recent years and it keeps getting more and more important as the whole world and everything in it gets connected to the Internet. For administrators, networks have become and are still becoming more and more complex when personnel brings their own devices (BYOD) into organizations network and virtual machines (VMs) and containers are everyday technologies. Adding Internet of Things (IoT) and Industrial Control Systems (ICS/SCADA) to the equation, the workload keeps getting higher and the understanding of the network and its state becomes blurry. In order to maintain or regain understanding of the state of a network and everything connected to it, new tools need to be implemented.

This thesis introduces a system, which consists of two main parts: monitoring system and log analytics platform. Monitoring system retains the state of devices connected to the network and provides alarming capabilities. Log analytics platform combines functionalities of traditional log management system, Security Information and Event Management (SIEM) system, machine learning and anomaly detection capabilities. The log analytics platform is integrated to the monitoring system in order to send alarms and link the events to the devices in the monitoring system.

While the most common use-case of the system is to detect human errors, configuration mistakes and malfunctioning software, it also helps to detect and analyze information security threats and incidents.

1.1 Problem overview

A monitoring system is needed for getting the status of a device connected to the network. Overall status is a combination of multiple things, for example: is the device up, how are the metrics of it — both hardware and software, are the necessary services up and functioning correctly etc. In a servers case, interesting hardware metrics could be environmental sensors, Redundant Array of Inexpensive Disks (RAID) status and Power Supply Unit (PSU) status. Interesting software metrics could be file system usage, network interface traffic and status of software updates.

In network devices case (switch, router, firewall), interesting metrics are obviously network interfaces traffic in addition to hardware status. Typical monitoring systems uses Simple Network Management Protocol (SNMP) to query how much traffic has been going in and out from an interface. While this information is useful for generating nice looking graphs of the amount of traffic in both directions, it does not provide a way for a deeper look into the traffic.

In order to find out source and destination IP addresses, port numbers, protocols and the amount of data transmitted between them, firewall log or flow data is needed. In servers case, same approach applies — in order to find out what has really happened, logs needs to be interpreted. If a service goes down or crashes, monitoring system will trigger an alarm about it. But it may not reveal the reason why it happened. That is why logs needs to be collected, stored and analyzed.

1.2 Solution overview

The monitoring system is not only distributed but also multi-tenant by design. That has to be taken into account when designing log analytics platform so it will function the same way and keep the data hygiene in place. An organization may have multiple sites all over the world, and only the administrators of a specific location has to be able to see their own data and only their data. Global top-level administrators have to be able to see every site's data in order to manage the system in a centralized way.

The location of the data is another story. Monitoring data is located only on their respective sites and can be queried from the central site. The idea behind this decision is to be able to keep monitoring even though the network connection would not work outside the remote site. Also storing monitoring data in a centralized location does not provide any benefits over a decentralized architecture.

Log analytics platform is designed exactly in the opposite way: logs are centralized to the central site and can be queried from remote sites. The idea behind this decision is that when logs are immediately exported to a separate location away from the original sites, they cannot be modified afterwards (at least makes it much harder). If the network connection between the remote site and central site breaks, the remote site stores the logs on disk until the connection comes back up or the buffer is filled. In addition, maintenance work and securing the data is easier when having to deal with only one instance rather than multiple instances in all different sites.

Integration between the two systems is thought as one-way channel: log analytics platform generates alarms based on predefined rules and anomaly detection and relays them to the monitoring system, which links the source of the alarm to the correlating device in the monitoring system and triggers the alarm to the specified end-users.

1.3 Research scope and goals

This thesis provides means for any organization to build a cost-effective and feature-rich system in order to reach an understanding of what is going on in the network and devices connected to it. The system will cover monitoring, log management, SIEM functionality and anomaly detection. The base functionality can be built with freely available open-source software but to extend the functionality even further, some commercial components are introduced. Although some examples are provided in the appendices, configuration of the introduced systems and providing detailed information about them is outside of the scope of this thesis.

1.4 Structure of the thesis

The rest of the thesis is structured as follows: Chapter 2 discusses the background information about situational awareness and how to reach it, the monitoring system is introduced in Chapter 3 followed by Chapter 4, which presents the log analytics platform. In Chapter 5, the system architecture and chosen design is described. Next, Chapter 6 discusses extensions and alternatives to chosen software. Finally, Chapter 7 presents conclusions and what more could be done in the future.

2. SITUATION AWARENESS

”Situation awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future.” [1]

This chapter describes what does Situation Awareness (SA) mean, what does it consist of and how to utilize the knowledge for defense. The chapter will also cover the idea of storing and analyzing logs.

2.1 Overview of situation awareness

Generally SA in scope of cyber defense consists of seven aspects: [2]

- Be aware of the current situation.
- Be aware of the impact of the attack.
- Be aware of how situations evolve.
- Be aware of actor behavior.
- Be aware of why and how the current situation is caused.
- Be aware of the quality of the collected SA information and the knowledge-intelligence-decisions derived from these information items.
- Assess plausible futures of the current situation.

The idea of SA in this thesis, is to give a complete overview to the whole computer system based on the previously listed seven aspects. Different views must have a very high abstraction level in order to see and understand instantly the state of the system. If there is something wrong, the administrator can drill down into the details, single lines of log (for example a TCP session) and historical monitoring data being the lowest level of abstraction in this system. The highest level relies heavily on visualization, in western world green means good and red means bad. If a service is down, the administrator is shown a big red notification. Graphs, charts and maps must be utilized in order to present the data in the most readable and easy to understand format.

SA in cyber security context includes the ability to detect, prevent and react to ongoing malicious activities. It can be achieved by collecting relevant data from multiple sources, such as network devices, servers and applications. This type of data can be parsed to standardized data structures, enriched with useful information and grouped together with other related data in order to visualize what is going on in the system.

SA is critical to every organization which has any type of network, servers or equipment to maintain. It not only provides an insight to the system in order to detect information security issues, but also acts as a support tool for end users. An example could be that if an end user is trying to enter his/her credentials multiple times without succeeding, the password is forgotten and if enough tries, the account gets locked. IT personnel can react to this type of situation immediately rather than letting the end user get frustrated and receive an angry email.

Two types of approaches can be used for monitoring the system and reaching situation awareness: monitoring (and archiving system state and events) and log management. In case of monitoring, the data is collected from different sources (monitored devices and services), evaluated if something needs to be done (alarming) and stored for later usage. In case of log management, logs are collected from different sources (devices and applications), parsed and enriched, stored and later on analyzed to detect interesting events. It is important to understand that while interesting events can be detected in the pipeline (while parsing the logs), an event can also be formed from multiple occurrences of a single event.

2.2 Log management in situation awareness

The importance of log management in situation awareness becomes very clear when thinking how can one really find out what has happened. If an application cannot provide proper logging (useful things gets logged in well-defined format) it is quite hard to find out what has happened (if an application misbehaved and crashed, someone gave malicious input, simple configuration error etc.). Logs are the most straight forward and common way to write output of what happened. For example, in 2015 TV5Monde was attacked and critical infrastructure was compromised. [1]

[1] https://static.sstic.org/videos2017/SSTIC_2017-06-09_P09.mp4

By having a centralized logging server collecting network traffic and server logs, the incident response was able to get to the bottom of things quickly. [2]

Log management can be handled in multiple ways: no log management, dumping to disk, central syslog server and proper log management system. In the first case, when something happens, first the administrators have to find out how a system writes logs (if it does), where are the logs (if there are any) and how to interpret them. This is obviously the worst approach.

The second approach is to rely on default configurations in everything. Usually applications and operating systems write logs to disk and there are some defaults (or not) how to handle rotation. Administrators knows where the logs are, have sometimes checked them if there has been an issue, but really don't have an idea about rotation and retention. The first and second approach suffers from the same downside: logs are seen only if there has been a critical error that someone has noticed. It takes time to find the right log and interpret it correctly, if it even exists.

The third approach is to have a centralized Syslog server where all the logs are gathered in one place and grouped in one way or another. The logs don't need to be searched from every location individually as they are in one place, so finding out what happened — even when involving multiple systems — is much faster than without centralized storage. All three aforementioned approaches suffer from the same downsides: Querying can be time consuming (or in some cases next to impossible) task and there is no event intelligence or analysis.

The fourth approach is to have a proper log management system in place. In this model, all the logs are gathered in one centralized place, parsed for easy querying and rotation and retention is defined strictly. Analysis can be done for the logs in order to detect anomalies and events and possibly alarm the administrators.

2.3 Information collecting methods

Logs can be typically gathered in two ways: write to a local file or send to local or remote logging server, typically Event Log for Windows and Syslog for Linux and Unix-based operating systems. In order to gather the logs into one centralized place, log files needs to be read and send to remote server.

Typical mid-sized organization could have:

[2]https://blog.comae.io/lessons-from-tv5monde-2015-hack-c4d62f07849d

- Firewalls, network switches and routers
- Server hardware
- Storage equipment
- User directory (LDAP) / Certificate authority (CA)
- DHCP and DNS server
- Email server
- Backup server
- Application / File / Database servers

The list can also be expanded with industry-specific data sources, like automation devices such as Programmable Logic Controllers (PLC) or Internet of Things (IoT) nodes. For these type of devices it is not necessarily useful to collect every type of log, but to monitor them in other ways. Typically IoT devices produce massive amounts of data, and malfunctions or breakages are not fatal, they are even expected. The most important thing is to receive the measured data and store it.

Of course many (if not all) of these services and equipment could be bought as a service, there are arguments for and against both approaches. While having everything on-premise can provide better value as a long term investment, administration and maintenance load is higher for the staff. Security-wise subscribing a service puts the maintenance load to the provider, but there can be concerns about data location, access and confidentiality.

Access logs have the information which user has tried to access monitored resource. An attempt can be successful or not, both are equally important. It depends a lot if 'who' can be identified as a individual, because the log line could include: a user name, an IP address (dynamic or static), a MAC address, a computer name or none of those. If a log line has a user name, it can be a shared account. If it has a dynamic IP address, DHCP logs have to be stored also in order to trace the computer. A MAC address can reveal the computer, or a network device, but can be easily spoofed.

DNS logs contains information about DNS queries and replies. They play an important role when trying to trace what has happened. The interesting parts are: client IP address (who made the query), request (what was asked), servers IP address (who replied to the query) and response (what was the content of the reply). Sometimes it is also useful to know which DNS server replied to the query, for example if it is not clear if the replying DNS server is a valid one or malicious.

It is extremely important to monitor login attempts (both remote and local) and changes in both configuration and policies. This is the situation where something has already gone wrong, the last call to realize that someone is in the system who does not belong there. This type of information can be found from authentication and audit logs. To prevent an attacker from reaching this state requires monitoring access logs, authentication logs, audit logs and traffic logs.

Authentication logs have the information about which account was used to log in, from where the login event came from, how (local, remote, RDP, VNC etc.) and when. Usually organizations use centralized user directory (for example LDAP) for storing and administering users and groups, in addition to a local administrator account in case of a communication issue with the user directory server. This is a good practice generally, but it is important to realize that monitoring both local and remote authentication methods are necessary in case of system failure (for example, communication channel between the system and LDAP server is broken). Authentication logs should always include at least: which credentials were used, what was the source IP address, if there was a process handling the authentication, which process was it and when did the authentication happen (date and time).

Furthermore, in order to track down who made a change, what was the change and when was it made, it is important to collect audit logs. This is extremely important for critical systems such as a user directory server, because someone could create another administrator account or change a policy to allow something malicious to happen.

Finally, traffic logs provide verbose low-level information which can be used to track which computer communicates to where, how much and when. Traffic logs can be received from network devices in Syslog format, any vendor-specific flow-format or IPFIX. Flow data includes at least source and destination addresses, the amount of traffic and time period. Traffic logs are very useful for example to track malicious usage of a resource and detect viruses and malware.

2.4 Preventing and reacting

Logging tells what has already happened by nature. For example, an anti-virus program quarantined a file on a computer or someone tried to login 10 times with incorrect password. Of course some events can be prevented by interpreting logs if there is a clear pattern or a message (disk is getting full), but monitoring system is designed to be proactive. It keeps current status of monitored devices and provides trends. Real life scenarios include: UPS battery is going to drain fully in 30 minutes, DHCP scope has only 5 leases left or a certificate is expiring in a week.

Monitoring system can also be configured to predict what is going to happen in the near future based on historical data and trends, what has happened before. For example, if a backup job is run every midnight, it causes a CPU load to spike for 10 minutes. If it happens every night, it should be considered normal and not to trigger an alarm.

Reacting to these types of issues varies a lot. There can be false-positive alarms which should not be acted upon, warnings which are more like reminders meaning that something is going to happen if not fixed (network interface has errors, UPS switched to battery power, 90% of RAM is used), or critical alarms which means that something is already broken (service has crashed, CPU utilization is 100%, network link is down).

Alarming can be done in many ways. Usually it is not necessary to get the information in real time if the alarm is not critical, so opening a helpdesk ticket or sending an email is sufficient. If the alarm is critical and administrators should be informed immediately, it is a good idea to send an SMS or push notification.

If the event is expected and is known to happen from time to time, scripting can be used to automatically react to alarms. Automating maintenance work by scripting monitoring system may not sound like a good idea (as it is not it's job and can become quite a complex system), but in some cases it is handy. A couple examples could be that if a service crashes, a script could be called to login to the server via SSH and restart the service. Or if someone tries to brute force their way in, a script can be called to add the attackers IP address to the ban table in the firewall.

2.5 Identifying critical systems and their valuable information

A number of data sources are needed in order to form an understanding of the situation in the computer system. It is very important to identify critical systems and what type of information is needed in order to secure them. Logs can be gathered from all mission critical systems, and they usually consists of authentication logs (someone tried to authenticate, success or failure), audit logs (someone changed configuration, what was changed, who it was and when) and traffic logs (who is transmitting to whom, what, when and how much).

The following table describes example products deployed in organizations and used in this thesis.

Table 2.1 *Example products used in this thesis.*

System	Example product
Firewall	SonicWALL
Network switches and routers	HP ProCurve
LDAP, CA, DNS, DHCP	Microsoft Windows Server
Application / File server	Microsoft Windows Server
RADIUS server	Microsoft Network Policy Server (NPS)

2.5.1 Accessing and shipping logs

Traditionally the most common way to transfer logs back-and-forth is to use Syslog. It is supported on all Unix-based operating systems, network devices, NAS equipment, applications and basically anything that can export logs.

Elasticsearch BV is a company behind popular open source products, such as Elasticsearch, Logstash and Kibana. They have also developed a protocol, which provides an alternative to Syslog, called Beats. While Syslog has seen many improvements and new features since the RFC 3164 [3] from 2001 like reliability (TCP) [4], security and privacy (TLS) [5] and authentication and integrity (signing) [6], Beats adds back pressure control with efficient window-size reduction in order not to overload the receiving Logstash instance.

At the time of writing, there are six officially supported Beats available: Packetbeat, Filebeat, Winlogbeat, Metricbeat, Heartbeat and Auditbeat. Packetbeat captures network traffic from the wire, Filebeat reads log files, Winlogbeat reads Windows event logs, Metricbeat collect metrics from the operating system and services running on a server, Hearbeat periodically checks the status of services and determines whether they are available and Auditbeat audits the activities of users and processes on systems. Community provided — but officially unsupported — beats are also available.

In this thesis, access logs are gathered from Linux servers with Auditbeat (which uses Linux Audit Framework), application logs with Filebeat, Windows server logs with Filebeat and Winlogbeat. Authentication logs are gathered from network devices, servers and applications via Syslog. Filebeat is used for Linux servers, Filebeat and Winlogbeat for Windows servers. DHCP logs are written to disk, so Filebeat is used to ship those, and Packetbeat is used to ship DNS logs.

2.5.2 Parsing different types of logs

Traffic logs are gathered from firewalls, switches and routers via Syslog. For firewall presented in Table 2.1, its Syslog format looks like this:

```
<129> id=firewall sn=ABCDEF123456 time"2017-05-12 14:44:24"
    fw=1.2.3.4 pri=1 c=32 m=809 msg="Gateway Anti-Virus
    Alert: Eicar-Test-Signature (Trojan) blocked." pktdatId
    =6419212868826169360 n=13 src=213.211.198.62:80:X1 dst
    =192.168.123.123:58842:X3 proto=tcp/58842 fw\_action="NA
    "
```

When parsing this log line, it is straight forward to apply a key-value filter with delimiter being "=". It is useful to rewrite cryptic field names, in this example field "m" is for event id, so it can be rewritten as "event_id".

Some events include more information than others. For example, message stating that TCP connection has been closed also includes how many bytes were sent and received:

```
<134>id=firewall sn=ABCDEF123456 time="2017-09-06 15:36:47"
    fw=1.2.3.4 pri=6 c=1024 m=537 msg="Connection Closed" f
    =11 n=24244948 src=192.168.123.123:54086:X0 dst
    =104.244.42.66:443:X1 proto=tcp/https sent=2195 rcvd
    =5287
```

This is useful when searching for "top talkers", meaning which device has sent and received the most data in a time unit. Source and destination fields (src, dst) contains IP address, port number, interface name and VLAN id. But not all messages use this format, some might include only IP address, or port, or network interface, or any variant of these. This makes parsing the fields a bit tricky, but it makes sense to parse those fields to new fields in order to query and visualize the data in much greater detail. For example, src field is parsed into src_ip, src_port, src_if and src_vlan. Same for destination field. Then, geoip lookup can also be applied for IP address fields in order to find out where the sender and receiver are located geographically.

If the IP address in LAN is in DHCP range, DHCP logs are needed in order to find out which computer it really was. DHCP server log mentioned in Table 2.1 looks like this:

```
11,12/05/17,14:38:47,Renew,192.168.123.123,lkolaptop.intra.
    company.tld,ABCD12345678,,2442412981,0,,,,0
    x4D53465420352E30,MSFT 5.0,,,,0
```

With this log line, it can be said that a computer named "lkolaptop.intra.company.tld" with MAC address "ABCD12345678" renewed its DHCP lease and is using the IP address which was the destination of aforementioned virus detection.

The device can be physically traced with RADIUS server log. The log from RADIUS server mentioned in Table 2.1 looks like this:

```
"AD01","IAS",09/06/2017,08:12:29,1,"lassi.kojo@company.tld
    ","INTRA\\lassi.kojo","78-48-59-CF-92-80","F4-0F-24-21-
    F2-C1",,,"CN38F2D7Z2
    ","192.168.223.104",924,0,"192.168.223.11","
    MSM730LspTalo",,,19,,,2,5,"Secure Wireless Connections
    ",0,"311 1 192.168.222.245 08/11/2017 06:51:59
    832",,,,,,,,,"9f69318e-000003c3",,,,,,"78-48-59-CF
    -92-80-F4-0F-24-21-F2-C1-59-AF-83-C0-00-07-A4-C2
    ",,,,,,,,,,,,,,,,,,,"Secure Wireless Connections",1,,,,
```

After parsing and enriching this log line, it becomes clear that the device was using EAP-TLS as an authentication method, connected to access point named "MSM730LspTalo" having IP address 192.168.223.11, via Wireless network. Packet-Type is Access-Request and with session id: 9f69318e-000003c3 the Access-Accept packet can also be found quickly.

2.6 Handling the log data

The log data needs to be handled in order to parse it, enrich it with new useful information, standardize the structure and to analyze and visualize them. The original log line should also be stored with the enriched and structured information, because parsing can always fail. In case an authority asks for logs, they want raw logs which are not handled in any way.

Enriching logs can be done in multiple different ways. The following is used in this thesis: tagging, translating and mutating. Tagging can be used to differentiate the original source, application and format for a log. It is very useful when querying related data from multiple sources. A query can be made, for example:

```
tags: "lan"
```

in order to get all the logs from LAN, or

```
tags: "database"
```

in order to get all the logs from database servers.

Translating is used to translate cryptic values (for example, numeric or hexadecimal) to human readable strings. One example could be to translate the authentication source from a RADIUS log record of the product mentioned in Table 2.1, did it come from wired or wireless network:

```
dictionary = [
  "15", "Ethernet",
  "19", "Wireless" ]
```

Mutating is generally used to add, edit or remove data in the log record. It can be used to remove useless data in order to save disk space and memory as there is no need to store and index these values. Mutating can also used to add a field to an event in order to identify where did it come from and refine values in fields, strip characters, split events and fields to new ones and replacing field values with new ones.

Parsing different log formats in order to get the data in standardized format can be quite tricky. Even though there are some standards in logging (for example Syslog), vendors don't always follow the exact standard and they might add their own vendor-specific information. Windows operating system has it's Event log, Linux has multiple formats and every application either uses a logging facility provided by the operating system or defines their own format. In order to get the data into the database, parsing is needed for each log format.

Sometimes one can get away with simply applying key-value (kv) filter, which specifies log row as a series of key-value pairs and their delimiter. But most of times this is not the case, especially when it comes to 3rd party application logs.

A single log line is not usually an event (even though it might be), but an event can be formed from multiple lines of the same log or multiple different log sources. Once the monitored events are defined, it is possible to define what is normal operations and what is an anomaly and this type of events needs to be handled accordingly (for example, alert). It is important to remember that not all anomalies are malicious and malicious activities can be hidden in the normal traffic too [7].

Some events are informational and should not trigger an alert at all, but only to be listed somewhere in sight. Some events should trigger an alarm and it should be defined who to alert and by which means. A typical example could be: All administrators, via email. Or if a critical service goes down, everyone could be alerted via email and SMS.

Trying to interpret all types of different log formats, from different systems and to combine the relevant information together is a very time consuming task. For example, if a device is connected to a wireless access point and is doing something suspicious, in order to find out what is going on and trace it, at least four different logs are needed: NPS, DHCP, DNS and traffic log (either from a switch or firewall). NPS log is used to find out which access point the device is connected to (trace), DHCP log correlates the IP address to MAC address, DNS log tells which DNS queries were made and traffic log tells destination IP address and port, and the amount of data transmitted. A proper log management system is needed to achieve this type of functionality.

3. MONITORING SYSTEM

This chapter describes a monitoring system. It is a proactive system which keeps track of real time status about all the target systems and devices configured to be monitored. The status and received events are stored for later usage, and graphs are drawn based on values of different services. The monitoring system can receive and parse logs, even alarm based on log lines, but it is not a place for long term storing and definitely not for fine-grained analysis. That is what a proper log management system / SIEM is for.

Every system administrator needs some type of monitoring system in order to have an idea what is going on in their infrastructure. There are lots of monitoring systems in the market targeted to every kind of need, free or commercial, open source or propriatery.

Monitoring systems can be divided into two types: State and event based monitoring. State based monitoring means that the status of the target device is decided based on the current state of the target when it is asked or received (depends which way the monitoring works, is it pull or push). An example of status could be: ”Interface X1 is up”

Event based monitoring means that the monitoring system monitors events, something happens uniquely at some point of time. Events can be monitored from log files or received for example via Syslog. An example of an event could be: ”Administrator logged in at 04:00”.

3.1 What to monitor?

In order for monitoring system to be useful, it is needed to define what is important to monitor. These could be:

- configuration mistakes
- configuration changes

- license expirations
- CPU, network traffic and disk load spikes
- temperature and humidity changes
- hardware failures
- host or service downtimes
- UPS battery self-tests
- storage over-provisioning
- mail queue is increasing

In this thesis, Check_MK[1] is used as a monitoring system.

Check_MK is available as an open source product but the company behind it, Mathias Kettner GmbH, has also developed commercial add-ons which improves the performance and introduces new features. The architecture of Check_MK is visualized in Figure 3.1, in which boxes with blue background are included in all versions, boxes with yellow background have improvements or are exclusively available as commercial add-ons and grey boxes indicates external systems not related directly to the monitoring system.

3.2 Distributed monitoring

A monitoring system traditionally consists of a monitoring server and devices which will be monitored. The server polls the status of monitored devices and stores the results for graphing, alerting and archiving. They can be located in the same network, or reached remotely over one or multiple hops via SSH, VPN or any other secured communication channel, but the monitoring logic, administration interface and data storage are always centralized on one server.

Check_MK provides a flexible distributed architecture pictured in Figure 3.2. Instances can be spread to remote sites (also called slaves) and centrally managed from local site (also called master). It enables the decentralization of the whole monitoring system into separate locations but the instances functions as autonomous systems. This is possible because Check_MK uses their Livestatus interface to communicate between instances.

[1] `https://mathias-kettner.com/check_mk.html`

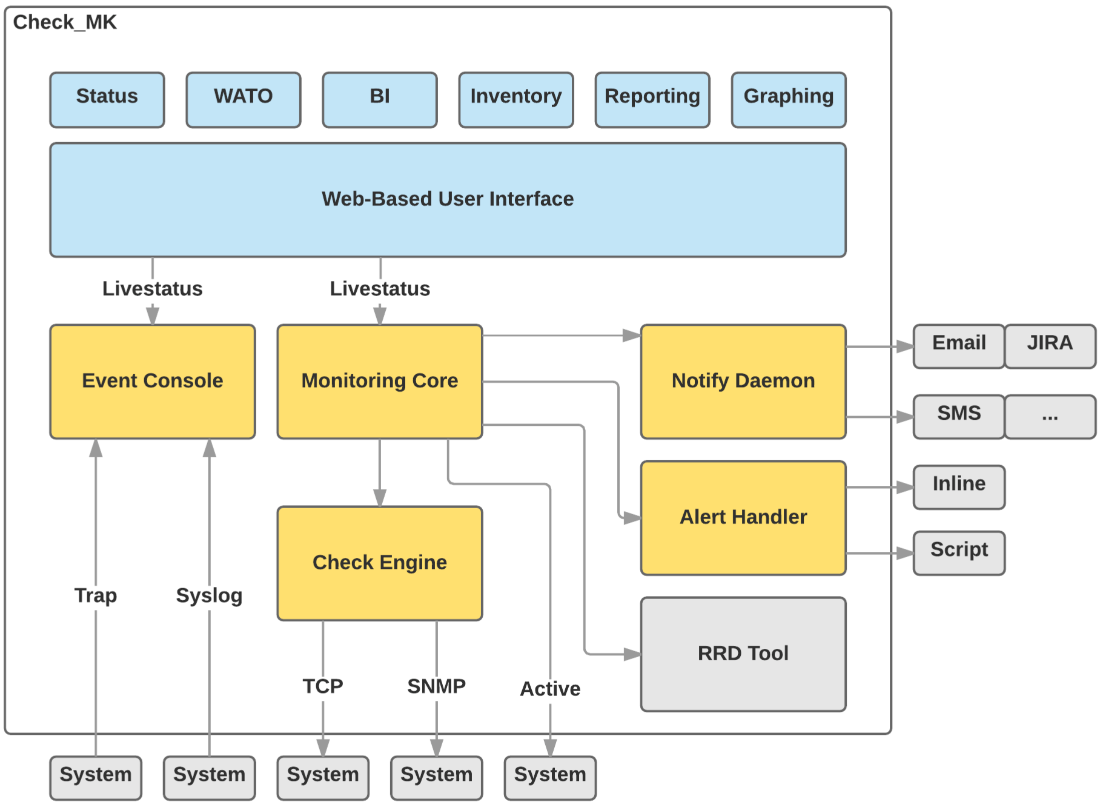

***Figure* 3.1** *Check_MK architecture. Components in blue background are included in all versions, components in yellow background may differ between the editions and components in grey background are external systems. Those marked with "System" label can be any device or software which can communicate using the protocols presented.*

3.2.1 Livestatus

Livestatus is an interface for querying monitoring data and executing commands, which is integrated into the monitoring core. Check_MK's user interface uses it internally to communicate with local and remote instances and is able to combine data from multiple instances into one centralized view. TCP socket can also be opened for external applications in order to provide an integration interface to and from external systems. [2]

Livestatus provides an in-depth API to the whole monitoring core. It is very lightweight, does not produce any IO-operations and generates low amount of CPU load as it runs on RAM. It can be used over both, Unix and TCP socket and it uses a query language called LQL (Livestatus Query Language) which syntax resembles HTTP. The query structure is similar to SQL, for example all services with current state being 2 (critical) with columns: host name, host description and host state,

[2]`https://mathias-kettner.com/cms_distributed_monitoring.html`

can be queried like this: [3]

```
GET services
Columns: host_name description state
Filter: state = 2
```

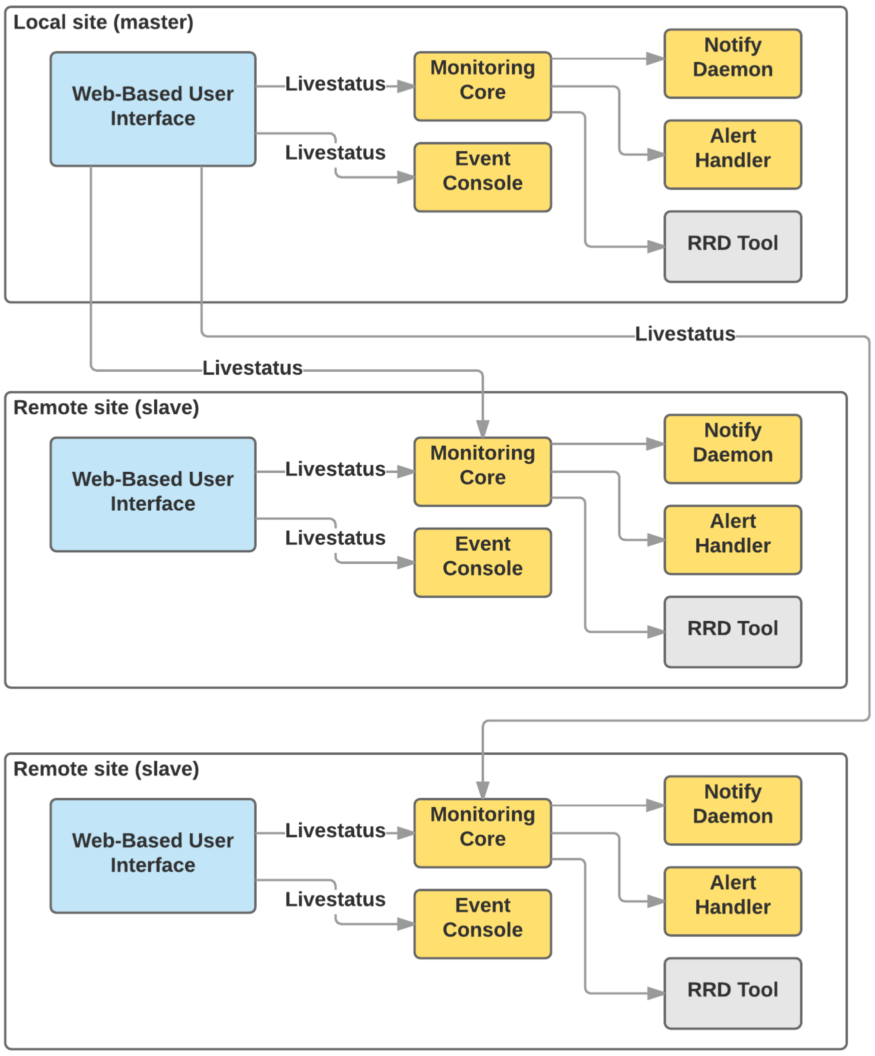

Figure *3.2 Distributed Check_MK architecture. Components in blue background are included in all editions, components in yellow background may differ between the editions and components in grey background are external applications.*

3.2.2 Benefits of distributed monitoring

Distributing the monitoring logic and data storage to multiple sites introduces lots of benefits in the following areas:

[3]`https://mathias-kettner.com/checkmk_livestatus.html`

- Redundancy
- Performance
- Security and privacy
- Simplicity

Redundancy is improved by distributing monitoring to multiple sites, so that network outages won't break monitoring and none of the sites depend on any other site. Local administrators won't see any difference even though communication between the master site and slave site wouldn't work.

Latency is smaller when querying results, because monitoring happens on every site locally, so the results (status data, events) are already retrieved from end devices. The only thing the master site has to do is query the status from the slave site and it will get a response almost as fast as the network operates. The monitoring itself does not generate any traffic between the sites, except possibly notifications if configured so.

Performance is increased not only because of less network traffic has to be generated, but processing is distributed over multiple computers. All the sites must take care of their own environment and only their own environment. While the master site can contribute to monitoring activities, it can also function as a centralized view to the system and would just query all the data from slave sites as shown in Figure 3.2.

Security and privacy improves as every site has only their own data. Slave sites are not talking with each other, so there is only one channel which needs to be secured, and that is between the master and slave sites.

Administering is simpler as every site sees only their own environment. This also applies to rule-sets. If everyone would see all the rules in the system, it would become quite complex and hard to manage over time.

3.2.3 Concerns of distributed monitoring

Because every site has their own data, all sites must be backed up separately. Notifications can be handled in a centralized or decentralized way. Both have their pros and cons.

In a centralized setup, there is no need to configure email servers or anything else to the slave sites other than forwarding notifications to the master site. If SMS is used, there's a need for only one SMS modem (in this thesis, this was the main reason for using centralized notifications). Notifications can also be relayed to other systems such as ticketing system.

Decentralized setup is simpler to construct as nothing needs to be relayed elsewhere and notifications work even if there is an issue with outside network connection (master cannot be reached).

Check_MK Enterprise Edition has an add-on called mknotifyd, which is a notification spooler. It allows both synchronous and asynchronous delivery of notifications and forwarding to another site. If forwarding is chosen, the connection can be formed either way, from master to slave or from slave to master. It uses generic TCP connection for reliable delivery.

3.3 Event console

Event console is a fully-integrated add-on created to complement Check_MKs monitoring logic to add events from external systems. It is a simple way to connect all types of systems to it because Syslog is very widely supported all around the enterprise computing hardware industry: servers, switches, firewalls, network attached storages etc. It processes Syslog messages, SNMP traps, Windows Event Logs and any other type of log files and other events of an asynchronous nature. [4]

When a message arrives to event console, a pre-defined rule-set is evaluated. If any of the rules apply to the message, an event is created based on the rule. Rules can also be configured to send notifications, trigger actions and rewrite text and attributes of a message. Events can be linked to a host in the monitoring system so that the event and notifications go to the right persons. Rules can also be configured to cancel previously created events.

Events sent to Event Console does not show up if no rules have been defined or if none of the events hits any rules. This is useful if a device is set to send logs to remote Syslog server, for example Event Console, and only critical events should be shown.

The event console can also be distributed. This makes sense, as both Syslog messages and SNMP traps are usually sent via UDP, and it is not always trivial to forward

[4] `https://mathias-kettner.com/cms_distributed_monitoring.html`

them reliably to the master site. This setup also supports the idea that every site should handle their own data, and it could be queried when needed.

3.4 Alarming

Check_MK uses rule based notifications for alarming. Using this feature, the administrator can define rules to determine what type of events causes alarms. Possible event types are: [5]

- ”a change of state (OK $\rightarrow$ CRIT)”
- ”a change from a steady to an unsteady (flapping) state”
- ”the start or end of a planned downtime”
- ”the confirmation of a problem (acknowledgement) by a user”
- ”an event arising from a manually-triggered notification command”
- ”the execution of an alert handler”
- ”an event passed for notification from the Event Console”

The rule based notification system in Check_MK is extremely flexible. The administrator can define a notification to be sent out for example, if the host name contains a string, service name contains a string, person X is contact person for host Y, output of a check contains a substring, every n^{th} event triggers a notification, same event has been triggered X times in the past Y minutes, just to name a few.

When an event matches any rule, the configured action will be triggered. An action can be basically anything as external scripts can also be called. Check_MK comes with a basic set of notification plugins: plain text and HTML email, forward to Event Console, push notification provider and SMS. More (currently unsupported) plugins can be found from their git repository [6] or one can easily write their own plugin.

[5] `http://mathias-kettner.com/cms_notifications.html`
[6] `https://git.mathias-kettner.de/git/?p=check_mk.git;a=tree;f=doc/treasures/notifications;hb=HEAD`

3.5 Event based actions

If an event triggers an alarm, Check_MK Enterprise Edition providers Alert Handler add-on to trigger any user defined action, which can be an inline or stored script. [7] One good use-case for this feature is to monitor a service which is known to misbehave from time to time. If it hangs or crashes, an action can be configured to Alarm Handler to log in to the web server via SSH and restart the malfunctioning service.

Another example would be that a VPN tunnel has died and cannot renegotiate itself back up, a script can be written to automate the rebuilding of a tunnel. A third example, if someone tries to brute-force their way into a system, a script could command the firewall to add the attacker's IP address to the ban table.

[7] http://mathias-kettner.com/cms_alert_handlers.html

4. STORING AND ANALYZING LOG DATA

This chapter describes how to parse, store, analyze and visualize log data. This chapter concentrates on log analytics platform, where logs are stored for long period of time and can be easily and quickly analyzed and visualized. Further in this chapter the integration between monitoring system and log analytics platform becomes clear when alarms are sent to the monitoring system in order to have the alarming configuration only in one place.

The log analytics platform is based on a product called Elastic stack from a company called Elasticsearch BV. It consists of Elasticsearch, Logstash, Kibana and Beats framework. Elasticsearch functions as a document store, Logstash collects, enriches and normalizes data, Kibana provides analytic intelligence and visualizations into the data and Beats provides a reliable way to ship the logs to the cluster over the network. The architecture of the stack is visualized in Figure 4.1.

4.1 Elasticsearch

Elasticsearch is a search engine built on top of Apache Lucene library. Both the search engine and the library are open source products. Like Lucene, Elasticsearch is also written in Java. It uses Lucene internally for all of its indexing and searching, but it aims to make full-text search easy by hiding the complexities of Lucene behind a simple API. [1]

In short, Elasticsearch is a distributed real-time document store where every field is indexed and searchable. It is capable of scaling to hundreds of servers and petabytes of both structured and unstructured data. Architecturally it is a server and it provides a RESTful API for clients. There are lots of existing web and command line clients written in multiple programming languages. [2]

The time between indexing a document and the time it becomes searchable is nor-

[1]`https://www.elastic.co/guide/en/elasticsearch/reference/5.5/_basic_concepts.html`

[2]`https://www.elastic.co/guide/en/elasticsearch/guide/2.x/intro.html`

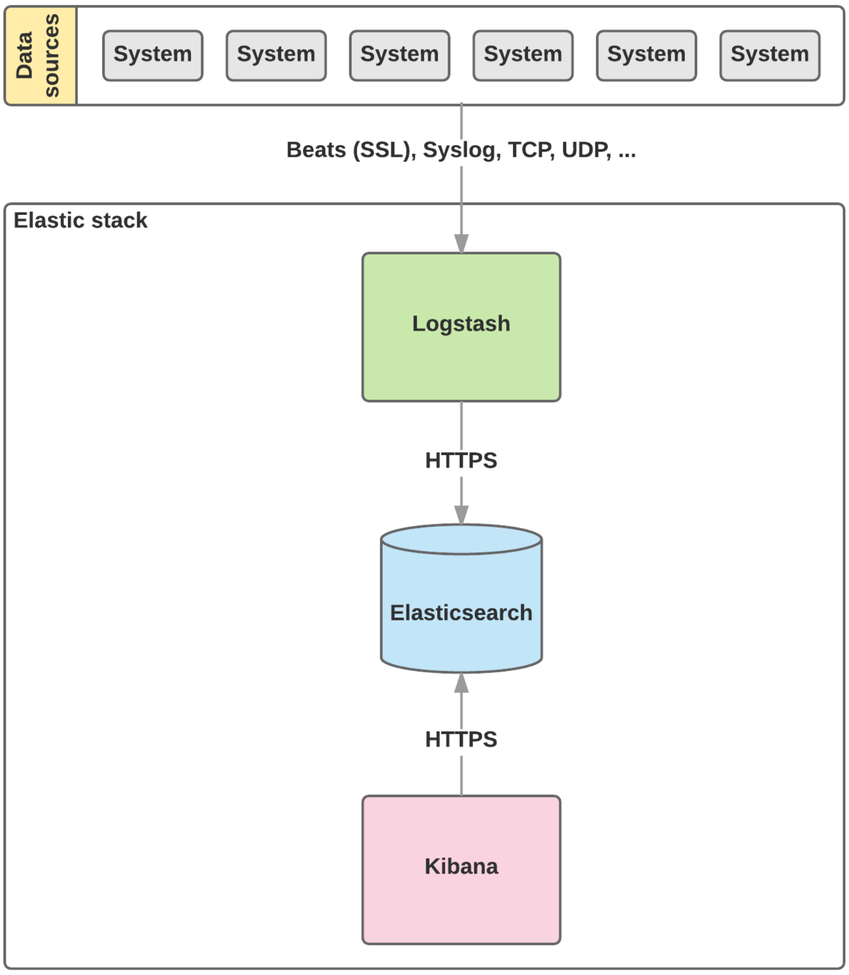

Figure 4.1 *Elastic stack architecture. External systems marked with "System" label can be any device or software which can communicate using the protocols supported by Logstash input plugins. X-Pack enables encrypted communication between all components and introduces authentication, authorization, users, roles and user directories.*

mally one second [3]. This means that Elasticsearch is a near real time search platform, which makes it a good choice for this type of system where incidents has to be detected immediately [8].

4.1.1 Node

A node is a single server that is part of a cluster. It stores data, and participates in the cluster's indexing and search capabilities. Just like a cluster, a node is identified by its name. Nodes have a capability to discover each other and look for a cluster to join. If they find the defined cluster, they will automatically join it and start

[3]https://www.elastic.co/guide/en/elasticsearch/guide/2.x/intro.html

working. If no other nodes can be found or there are no clusters to join, a node will start its own single-node cluster. [4]

Nodes can have one or multiple roles in the cluster: master-eligible, data, ingest, tribe, coordinating and machine learning (only with X-Pack).

A master-eligible node can control the cluster and a node can become a master node if elected. A data node stores data (shards) and can perform data related operations like searching and aggregating. An ingest node is able to transform and enrich a document before indexing in a similar way how Logstash functions. A tribe node can be used to connect multiple clusters to each other and perform operations across all of them. A coordinating node routes clients requests to nodes storing the requested data, gathers responses and returns the results to the client. A machine learning node can perform defined machine learning jobs. [5]

4.1.2 Cluster

A cluster consists of a single or multiple Elasticsearch nodes, which may — but not necessarily — have different roles. A cluster has a name and it is a unique identifier for a particular cluster: Nodes can be configured to join a cluster by its name.

Scaling can be achieved by adding more nodes into the cluster, or by replacing the current servers with more powerful ones. Increasing the number of servers is called horizontal scaling and switching to more powerful servers is called vertical scaling. Vertical scaling can provide better performance only until some point, but horizontal scaling provides true scalability: adding more nodes to the cluster increases performance and reliability by spreading the load (computational and storage) between all of them. [6]

Because Elasticsearch is designed to be distributed, it handles scaling and high availability automatically and clients does not have to care about the details. When a new node is added to the cluster, part of the stored data is automatically assigned to it in the optimal way to distribute load between the nodes and to maintain high availability. For example active data and its replicas should not be located in the same node. Queries gets routed automatically to a node which has the information in question, if the particular node which was asked the information does not have it.

[4] `https://www.elastic.co/guide/en/elasticsearch/guide/2.x/intro.html`

[5] `https://www.elastic.co/guide/en/elasticsearch/reference/5.5/modules-node.html`

[6] `https://www.elastic.co/guide/en/elasticsearch/guide/2.x/distributed-cluster.html`

A three-node cluster where all nodes are master-eligible and stores data is pictured in Figure 4.2

4.1.3 Indexes, types and documents

An index can be thought as an equivalent to a database in the relational database world. An index is a collection of documents that have similar characteristics. It is identified by its name and that is used to refer to the index when performing operations against documents in it.

Within an index, one or more types can be defined. A type is a logical category of an index whose semantics is completely definable by the administrator. In general, a type is defined for documents that have a set of common fields.

A document is a basic unit of information that can be indexed. Documents are expressed in JSON (JavaScript Object Notation) which is a ubiquitous Internet data interchange format. Although a document physically resides in an index, a document must be indexed/assigned to a type inside an index.

4.1.4 Shards and replicas

A single index can store, for example, billions of documents which take up a lot of disk space. This may lead to the situation where the size of an index exceeds the hardware limits of a single node, becoming bigger than available disk space or requesting information taking too long for a single node alone. [7]

Elasticsearch solves this problem by subdividing an index into multiple pieces which are called shards. When creating an index, the number of shards is defined. Each shard is a fully-functional and independent Apache Lucene -instance which can be hosted on any node in the cluster.

There are two types of shards: primary and replica shards. Primary shard is an active shard, which will intake new data as it comes. By increasing the number of primary shards, the maximum number of documents possible to store in an index can be increased.

Replica shards are copies of primary shards and new data cannot be added to them directly. In case a shard goes offline or disappears for any reason, a replica shard of

[7] `https://www.elastic.co/guide/en/elasticsearch/reference/5.5/_basic_concepts.html`

the lost primary shard is promoted to the new primary shard and it starts to intake new data, while the other replicas will start to replicate the new primary (if more than one replica shard is configured).

Replication is important because it provides high availability in case a shard or node fails and it allows to scale out volume and throughput because searches can be executed on all replicas in parallel. It should be noted that the number of replica shards can be dynamically changed anytime but the number of primary shards cannot be changed after the index has been created.

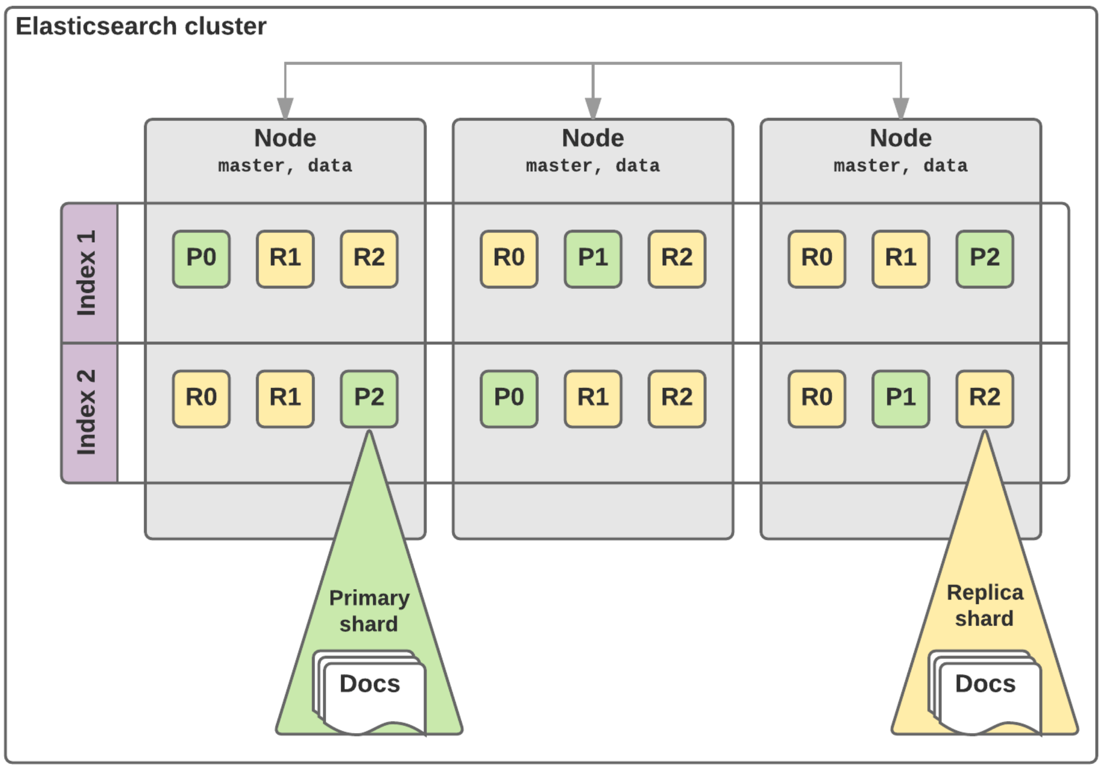

Figure 4.2 *Elasticsearch architecture. Three master nodes is the minimum recommended for a cluster (because of split-brain issue). Index is a cluster-wide concept, data (documents) is stored in shards and shards are distributed to nodes.*

4.2 Logstash

Logstash is a data collection engine and also an open source product. It has real-time pipelining capabilities which can be utilized with plugins. [8]

Logstash is used to collect, enrich, unify and normalize data from different sources and to save it to multiple different outputs. In this thesis, Elasticsearch cluster and

[8] `https://www.elastic.co/guide/en/logstash/5.5/introduction.html`

Syslog outputs are used. Even though Logstash was originally meant to process logs (as the name indicates), it can be used to process any type of events or messages. [9]

Logstash uses pipeline architecture visualized in Figure 4.3, and it has three different types of plugins: input, filter and output. They can be mixed together and used as many times as needed.

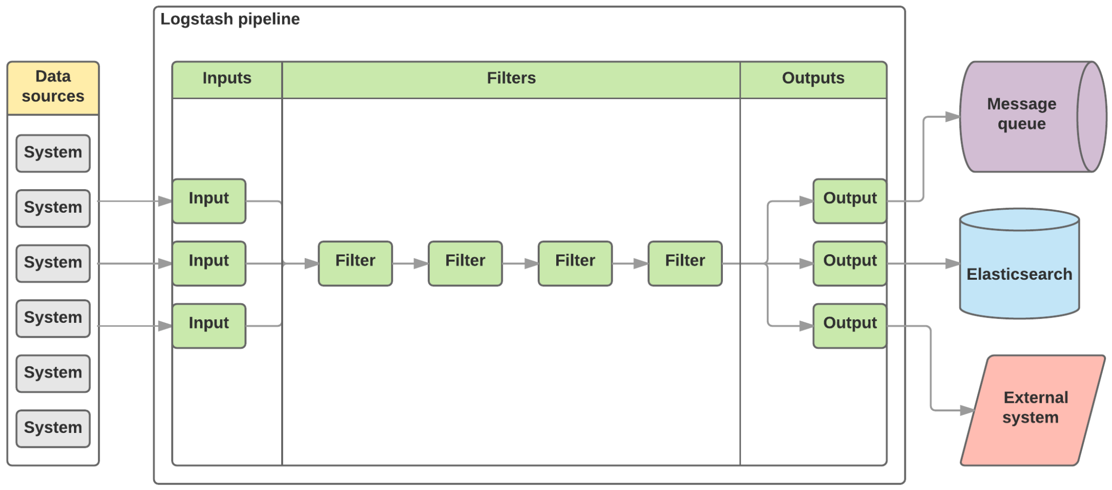

Figure 4.3 *Logstash pipeline. Inputs and outputs are executed in parallel, filters sequentially. External systems marked with "System" label can be any device or software which can communicate using the protocols supported by Logstash input plugins*

4.2.1 Inputs, filters and outputs

At the time of this writing, there are over 200 plugins made for Logstash. In small setups and homogenous environments, only a few of those are usually necessary, but the possibilities are almost infinite. These plugins consist of inputs, filters, outputs and codecs. In this thesis, the following plugins are used.

Inputs:

- beats - Receives events from the Elastic Beats framework
- dead_letter_queue - Reads events from Logstash's dead letter queue
- http - Receives events over HTTP(S)
- tcp - Reads events from a TCP socket

[9] `https://www.elastic.co/guide/en/logstash/5.5/introduction.html`

- udp - Reads events over UDP

Filters:

- csv - Parses comma-separated value data into individual fields
- date - Parses dates from fields to use as the Logstash timestamp for an event
- geoip - Adds geographical information about an IP address
- grok - Parses unstructured event data into fields
- kv - Parses key-value pairs
- mutate - Performs mutations on fields
- translate - Replaces field contents based on a hash or YAML file

Outputs:

- elasticsearch - Stores logs in Elasticsearch
- syslog - Sends events to a syslog server

Codecs:

- cef - Reads the ArcSight Common Event Format (CEF)
- json - Reads JSON formatted content, creating one event per element in a JSON array
- plain - Reads plaintext with no delimiting between events

4.2.2 Data resiliency

In case the data cannot be exported to the Elasticsearch cluster (for example, unparseable logline, Logstash crashing, network between Logstash and Elasticsearch cluster is down), there needs to be a way to store the data until the malfunction is fixed. Logstash provides two features which helps to prevent data loss in different cases: Persistent queues protect against data loss in case of network breakage and unexpected behavior, dead letter queues protect against unprocessable logs.

Both queues are stored on disk and they need to be specifically enabled as they are disabled by default.

Persistent queue works by storing every log line to disk as soon as it has been received by an input. This feature also works as a buffering mechanism to handle enormous bursts of log lines without needing separate message broker, such as Redis or Apache Kafka. In short, persistent queues provides an at-least-once delivery guarantee against data loss during either a normal shutdown or unexpected crash. When Logstash is restarted, it tries to deliver on-disk queues until it succeeds. When designing for maximum resiliency, it should be understood that not all input plugins support persistent queues. [10]

Sometimes Logstash cannot process an incoming log line. This can be caused by, for example wrong character encoding, parsing failure or mapping error. In this case, by default, Logstash will drop the message and move on. By enabling dead letter queues, Logstash will store all of these events on-disk for later processing. [11] Events can be read from the dead letter queue to Logstash with the ”dead_letter_queue” input plugin and processed through the Logstash pipeline.

It is important to understand that neither persistent queues nor dead letter queues protect against data loss if the amount of data exceeds configured maximum amount of space reserved for them.

4.3 Kibana

Kibana can be used to browse logs from different indices, generate visualizations based on the logs stored in Elasticsearch, monitor all Logstash instances and watch and alarm on pre-defined events if they are found from the log data. It is an analytics and visualization platform for Elasticsearch and is also open source project maintained by Elasticsearch BV.

Discover page provides a time-based view to the logs. Events can be queried by entering indexed field name and value, for example: ”event_id: 809” or full-text search can be used: ”Virus was detected”. Queries can be grouped together with boolean-operators so that only very specific and useful information can be found. This query can be saved and used as a filter for visualizations.

Analyzing a large log file is hard and time consuming. Visual approach facilitates the task significantly as the human brain is an amazing pattern-recognizing tool,

[10]`https://www.elastic.co/guide/en/logstash/5.5/persistent-queues.html`
[11]`https://www.elastic.co/guide/en/logstash/5.5/dead-letter-queues.html`

and it is said that ”a picture is worth a thousand log records.” [9]. Visualizations can change the patterns, sizes, colors, shapes and textures in order to group data. Visualization has lots of benefits, such as: [9]

- ”Answers a question”
- ”Poses new questions”
- ”Explore and discover”
- ”Support decisions”
- ”Communicate information”
- ”Increase efficiency”

An image can quickly answer a question without digging through multiple lines from different log sources. It usually poses new questions: Why is this machine communicating to this one”. Different types of visualizations provides new views to the same data, which triggers new thoughts. By combining data from multiple sources and presenting it graphically, visualizations support decisions and help with SA. [9]

Collecting logs from different types of systems, especially firewall traffic, generates huge amounts of data. Visualization makes it easy to understand large volumes of data, to expose critical events, anomalies and trends (periodical events). It is a high level tool for analyzing the large mass of data.

Kibana supports different types of visualizations, for example: maps, tables, pie and area charts, metrics and heat maps. In this type of system, example use-cases could be that maps are used for geographical data, metrics are used for the number of virus detections for the day and bar charts are used for logins and IDS events (time in x-axis and number of events in y-axis).

Visualizations are meant to visualize the stored data. It is better to create multiple visualizations for multiple different things rather than cramp a lot of different, but related, information into one single visualization. Dashboards are for collecting related information together. Dashboards consist of multiple visualizations and their function is to provide a comprehensive view to similar types of information. They are useful to keep open for example on info-display.

4.4 Reporting

Reporting in monitoring system is a commercial feature available in Check_MK Enterprise Edition. Same with log management system, it is a commercial feature available with X-Pack.

Check_MK's reporting can produce availability reports which also include received alarms from log management system. X-Packs reporting feature can include raw log lines and visualizations. Both reports can be scheduled and automatically sent via email.

4.5 Alarming with Watcher

Watcher is a feature of commercial expansion for Elastic-stack called X-Pack[12]. It can be configured to watch both the cluster itself and the events stored in the Elasticsearch. Configuration consists of four parts:

- Schedule, when to run a query and check condition(s).
- Query, what to search from the Elasticsearch.
- Condition, whether or not to execute the action(s).
- Action, what to do when condition is true, for example: send an email, open a support ticket or store results in an index.

Alarming is a bit different with this type of system than proper monitoring system by design. A traditional monitoring system generates alarms immediately when the state changes or an event happens (and it is detected). After that, it is stored for historical usage. With this type of log management system, the data is already stored in Elasticsearch and it needs to be actively queried in order to detect interesting events. Alarming could also be handled with Logstash immediately after receiving an event or multiple events, and before storing it to Elasticsearch, but this solution proved to be easier to configure and maintain. Also, detecting an event based on multiple separate messages is not easily handled with Logstash alone.

A challenge with this type of alarming is how to make sure one event is alarmed only once. If a query is run every five minutes and a query timeframe would be more than five minutes, duplicate alarms would happen. If a query is run every five

[12]`https://www.elastic.co/guide/en/x-pack/5.5/xpack-introduction.html`

minutes and a query timeframe is five minutes or less, not all the alarms would ever be triggered. One possible approach is to make sure an alarm would trigger only once is to create an ”acknowledged” field to every document which could possibly be part of an interesting event. Watcher piggybacks necessary data to identify a document in Elasticsearch cluster and when Event Console receives an alarm, it acknowledges documents it has received (sets ”acknowledged” field to true). This also makes sure that an alarm is not lost between Logstash and Event Console, or if it is, it will get sent again until it succeeds.

The acknowledgement approach was discarded later on because acknowledging messages in aggregations proved to be quite complex.

In this thesis, alarms are sent to Check_MK Event Console via Syslog protocol (Event Console is acting as a Syslog server, listening on TCP socket). Watcher is configured to send a HTTP POST to a Logstash instance, which is listening for incoming HTTP messages with ”input-http” plugin. Logstash tags the incoming POST messages with ”alarm” tag, reformats and outputs them with ”output-syslog” plugin to Check_MK’s Event Console. It is necessary to identify from which remote site (or local site) the host originating the message is, because if the originating host is on remote site and the alarm is sent to the local site, Event Console cannot identify the host from the remote site and the alarm either goes into the void or to preconfigured fallback address. The alarming process from log analytics platform to monitoring system is visualized in Figure 4.4. After receiving a message, Event Console is configured to parse the interesting parts of the message and reformat it to human-readable format.

4.6 Anomaly detection

While writing Watcher rules and defining critical events is a straightforward approach, the configuration can become complicated after a while. A couple of tens of rules is still fine, but hundreds or even thousands of rules is unsustainable in the long run. On top of that, it is very hard or almost impossible to cover all the interesting cases which should trigger an alarm.

To help with this situation, X-Pack provides a machine learning capabilities for anomaly detection. Machine learning — in this case — means that historical data stored in Elasticsearch cluster is analyzed and determined what is normal.

For example, in a normal situation administrators log in to a several servers every day during office hours. And if there is a night shift, during night time too. A

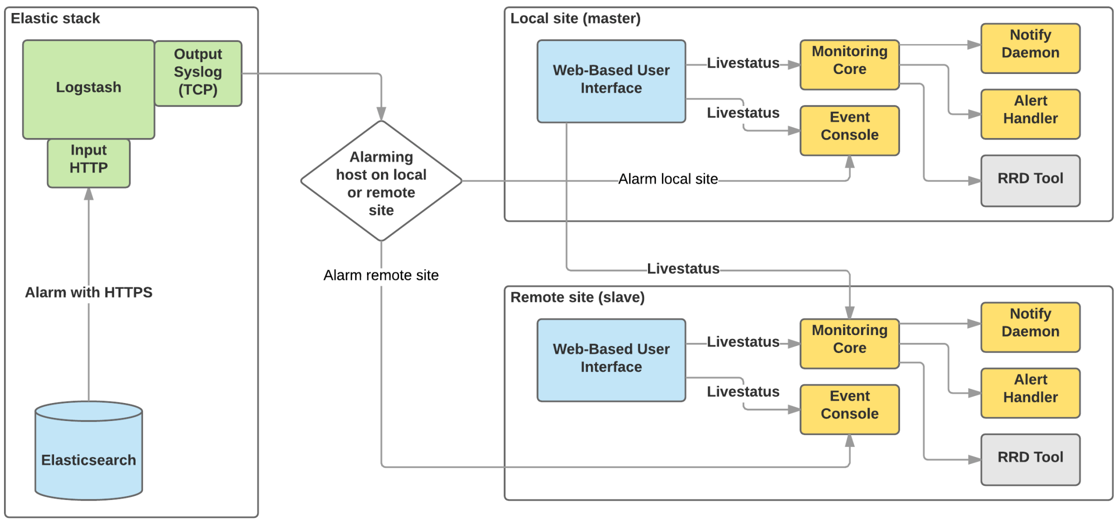

Figure 4.4 *Send alarm from Logstash to Event Console. If an alarming host is located on local site, send it to local site's Event Console via TCP. If on remote site, send it to remote site's Event Console via TCP. All alarms will be delivered from remote sites to the local site and alarms are sent from there.*

person working during office hours logs in during night shift is not normal, and should trigger an alarm. Or if a salesperson has traveled to another continent to do business, the first couple of logins through VPN should trigger an alarm, but the system should learn that this is the new normal (for now) and stop alerting on those logins. With the same idea in general, everything out of the ordinary should trigger some type of alarm and administrators should react and see if something needs to be done or not.

Anomaly detection is a valuable tool for spotting configuration mistakes, malfunctioning devices and end-user mistakes. While it also helps to identify typical attacks, it shouldn't be relied on too much. A sophisticated attacker who wants to penetrate the system understands that anomalies will be detected and will try to disguise the attack to look like normal traffic. [10]

5. SYSTEM ARCHITECTURE

A system like the one described in this thesis can be set up in many different ways. This chapter describes the final system architecture and explains decisions made behind the design.

5.1 High level architecture

High level system architecture is presented in Figure 5.1. Firewalls, switches, wireless access points, servers and applications store their logs into the monitoring and logging server located in their own site. Logs are transferred via VPN tunnel to the logging cluster located in master site (local site in Figure 5.1). Logs are analyzed and stored in the cluster and if interesting events appear, alarm will be triggered. After storing the logs, they can be queried from both local and remote site.

Monitoring instances poll the status and listen for incoming events via Event Console on all sites. The monitoring data is stored on each sites' own instance, and can be queried from master site (local site in Figure 5.1).

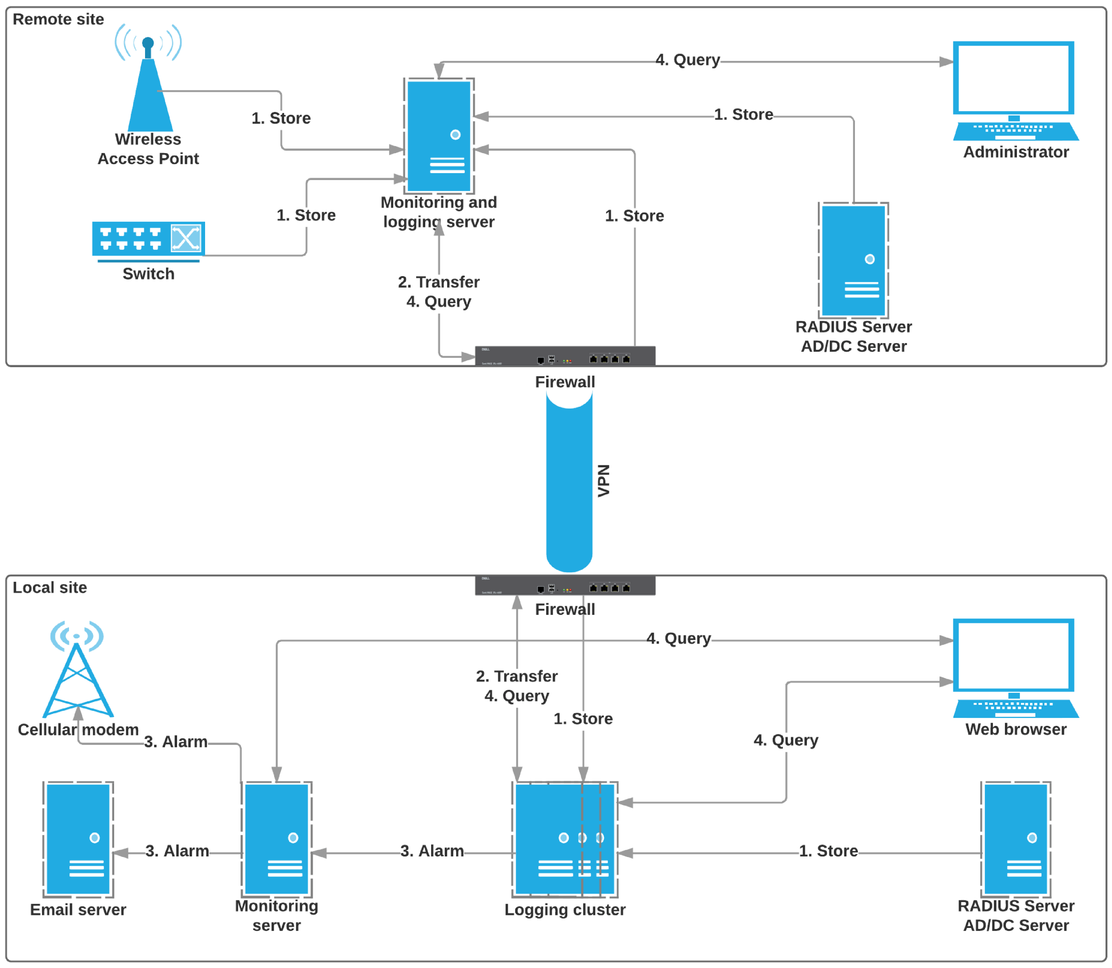

Figure 5.1 *Logical system architecture.*

5.2 Low level architecture

The low level system architecture is presented in Figure 5.2. Firewalls, switches and wireless access points are sending their logs over Syslog UDP, server and application logs are exported over SSL secured Beats protocol. Logs are transferred over HTTPS into the Elasticsearch cluster, where they can be queried with Kibana instances over HTTPS. If interesting event appears, alarming is implemented over Syslog TCP.

Monitoring instances communicate with Check_MK Livestatus, mknotifyd (Notify Daemon) and mkeventd (Event Console) and they use their own TCP sockets for communication between different sites. Event Console can also listen on UDP socket.

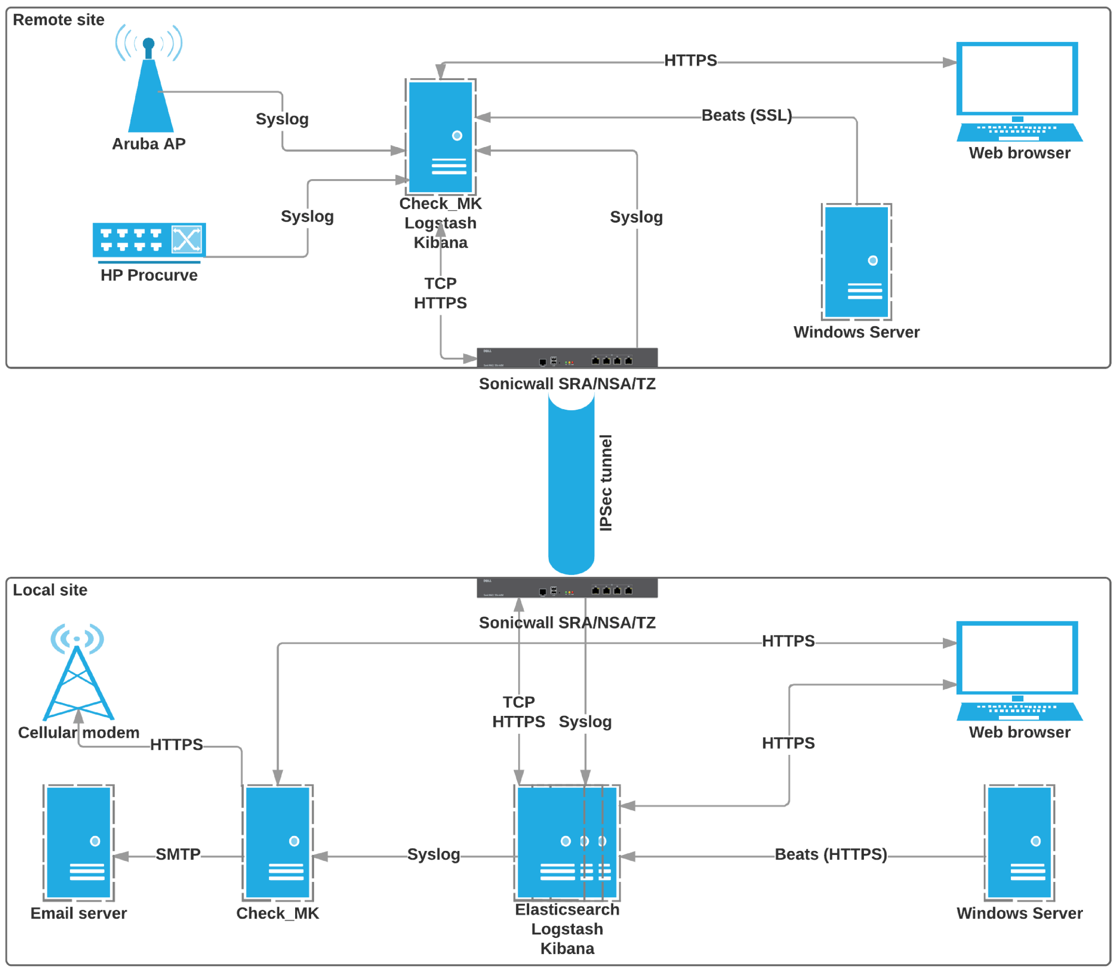

Figure 5.2 *Physical system architecture. Only monitoring system is communicating back and forth, logging system communicates only from remote site to local site.*

5.3 Securing communication channels

VPN tunnels are protecting the traffic between local and remote sites, for both monitoring system and log management system. This is to prevent publishing anything to the Internet directly, but just export a route from local monitoring instance to remote instance, and from remote instance to local monitoring instance and Elasticsearch cluster for log management. This makes the VPN tunnel dedicated to monitoring and logging traffic, it won't include any user generated traffic and hence makes it more secure.

End users can use the instance in their local sites to access monitoring and logging interface, but they won't be able to access the tunnel. Both Check_MK and Kibana interfaces are published to their sites LAN networks with HTTPS and the certificate is signed against their own CA's certificate.

It is worth noting that there are two types of HTTPS connections: Between Elasticsearch cluster and Logstash/Kibana, and between Kibana and end users. Communication between Elasticsearch cluster and Logstash/Kibana is secured with certificates signed against a dedicated CA created just for these connections. Communication between Kibana and end users is secured with certificates signed against sites own CA's. Usually CA certificates are pushed to the end users' computers via GPO (Windows Group Policy Object) so the trust chain should get formed without any user action.

NAT is configured so that the remote sites are shown with their public IP address to the master monitoring site and Elasticsearch cluster. Master monitoring site and Elasticsearch cluster and shown with their own private IP addresses through the VPN tunnel to remote site. This enables to replicate the same configuration to all sites without having to worry about overlapping subnets and IP addresses.

5.4 Securing stored data

It would be a disaster if any of the remote sites could see someone else's data. To mitigate this issue, X-Pack was acquired in order to gain support for different authentication providers, user and role management, TLS encryption throughout the whole system and document level security.

Different sites are differentiated by a prefix, so it is straightforward to define which indices should be accessed by which user/role. Every site has their own users, which are tied to their own role. A role is only allowed to read and access only their own data. No one should be able to modify or delete any data from the cluster, except cluster administrator account, which should be kept in a safe and used only for mandatory maintenance tasks. Every site has their dedicated Logstash user, which is tied to their own log exporter role. That role is only allowed to create indices with the sites' prefix and write in them.

Finally the whole system needs to generate an audit log. It includes all the operations in the cluster, for example which account read or wrote information, from which IP address and when. The audit log is exported outside of the cluster in order to keep it safe in case of a breach.

5.5 Alternative approaches

Securing Elastic stack is straightforward with commercial X-Pack supported by the same company behind Elastic stack. X-Pack was chosen to be used in this thesis mainly because it is officially supported, but there are other alternatives.

Search Guard[1] offers authentication, authorization, audit logging, encryption and multi-tenancy. The project is open source but some features are not available for free.

KEK (the High Energy Accelerator Research Organization) has developed their own plugin on top of Search Guard, called kibana-own-home [2] [11]. Elastic stack is also used in CERN (the European Organization for Nuclear Research) [12] and they have developed in-house plugins for securing it and enabling multi-tenancy. KEK's solution provides a different approach for multi-tenant Kibana than CERN's solution, as described in their paper [11].

ReadonlyREST[3] offers all the same features than previously presented alternatives, and also like them, is open source but more functionality offered with a price. ReadonlyREST relies on Access Control List (ACL) based approach, while X-Pack and Search Guard relies on Role-Based Access Control (RBAC) [13]. [4]

Elastalert[5] and Sentinl[6] are alternatives for Watcher. They both provide alerting on different types of patterns of interest from data in Elasticsearch and Sentinl also provides reporting capabilities.

5.6 Remote Syslog

Transmitting information over Syslog protocol is a bit tricky. The standard way is to use UDP port 514, only the newer RFC 6587 [14] brought standardized TCP implementation, even though there was already previously different types of implementations. It would be naïve to think that most of the devices and applications would support TCP transmission of Syslog, so it was decided to use centralized Syslog server to collect all the logs from different sources into one place.

In the early stage of this project, syslog-ng was chosen as a centralized Syslog server

[1]`https://floragunn.com`
[2]`https://github.com/wtakase/kibana-own-home`
[3]`https://readonlyrest.com`
[4]`https://cds.cern.ch/record/2261999`
[5]`https://github.com/Yelp/elastalert`
[6]`https://github.com/sirensolutions/sentinl`

because of its stability, extendability and popularity. It has the ability to open listening sockets for both UDP and TCP protocols. As this type of data is crucial, TCP should be used whenever possible in favor of UDP.

Designing the optimal implementation to maximize the reliability of gathering logs over UPD, securing sensitive plain-text information about the organization's computer system and making sure the availability is at an adequate level proved to be a laborious task. Monitoring data needs to be available for querying and log data has to be exported from remote site to local one over Internet.

The first design was to establish a VPN connection between the local and remote site, and relay the Syslog messages into the tunnel. This approach has a couple of drawbacks: VPN tunnels can timeout or if there are overlapping subnets, NAT has to be used.

The second design was to configure syslog-ng just to listen to incoming Syslog messages, enrich it with information about the location (origin) and monitoring hostname. After that, the message would be relayed to the Logstash. But monitoring audit logs from the monitoring instance is also required, and for that using Filebeat makes sense because of its features like congestion control, transport layer security and it is very easy to plug into Logstash as an input.

In the third design, syslog-ng listens for incoming messages, enriches them but writes them into a log file on a disk. Then Filebeat reads those log files and sends them to Logstash. This has a couple of advantages over the first and second designs: simpler architecture (one program does one thing), easier configuration (Filebeat is a bit easier to configure and manage as an input for Logstash) and reliability (if there is a network breakdown, logs gets written on the disk, and when the network connection recovers, the logs are transmitted to Logstash).

The final design changed things quite radically. Separate Syslog server was decided to be left out and instead Logstash will be installed on the remote site. This makes the configuration between all remote sites similar, Logstash also provides almost endless possibilities to handle different types of logs, the heavy lifting can be outsourced to the remote sites and there is no need for separate Syslog server or Filebeat when every remote site has their own Logstash instance. Logstash would use HTTPS to transfer logs into Elasticsearch cluster and there is already a VPN tunnel between the local and remote sites because monitoring systems uses it, so it was decided to utilize the same tunnel and avoid opening Elasticsearch to the Internet in any form as such (obviously it should only be opened against the remote sites IP addresses). Later on it was decided to also deploy Kibana to the same machine and have remote sites as autonomous instances totally separate from the local site except the Elasticsearch cluster, where the data is stored. This provides maximum resiliency against network outages and data loss.

5.7 DNS logging

First the idea was to use a firewall in order to log DNS payload, but it didn't have such a feature. When giving a bit more thought to DNS logging, it would not make much sense to use anything else but the DNS server itself to log DNS payload. Usually mid-sized organizations have their own internal DNS server to provide intranet DNS services and to query external names. Now if there would be a packet sniffer between LAN and intranet DNS server, the responder would always be the intranet DNS server, and not the one actually giving the reply. If the sniffer would be placed between intranet DNS server and Internet, it would not contain information about the original source of the query. Because of this, logging has to be located on the intranet DNS server. This way it is possible to get information about the original source and also the server replying from Internet. Basically what was asked and what was the reply, who asked and who replied.'

Trying to capture DNS logs from Windows DNS server proved to be far from trivial task. First, the whole feature is only available in Windows Server 2016, and a hotfix is available for Windows Server 2012 R2. This means that the approach is not generic but actually very limited. Second, it does not support logging to a file, but only acts as a provider to Microsoft's Event Tracing for Windows (ETW)[7]. This means, that it would be necessary to use a consumer application for ETW, because none of the beats supports it. While reading what others have done, like writing their own consumer applications [8] or using Tracelog[9], it became obvious that it would be an incredibly ugly hack to pursue this approach.

After giving it more thought, it was decided to install Packetbeat to the DNS server and ship DNS traffic to Logstash. Granted, it is far from ideal to install a packet sniffer (WinPCAP[10]) in a production server, but it is the least painful and the most generic solution, and could prove to be useful in other areas like logging and debugging LDAP connections.

[7]`https://technet.microsoft.com/en-us/library/dn800669(v=ws.11).aspx`

[8]`https://blogs.technet.microsoft.com/teamdhcp/2015/11/23/network-forensics-with-windows-dns-analytical-logging/`

[9]`https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/tracelog`

[10]`https://www.winpcap.org`

6. CONCLUSIONS

There are lots of commercial products and open source projects in this field. The price seems to be a culprit for many organizations not to get a proper log management / SIEM product [15].

Using open source software does not mean that it is in any way free or even cheap. Source code itself might be, but learning to use the software can take a lot of time and with this type of setup at least three servers are a recommended minimum. Elastic cloud (and other services providing Elastic stack) is also available if an organization does not want to invest in their own servers, but the data is so sensitive that it may not be a good idea to export the logs into a public cloud. There can also be a problem with throughput and bandwidth it the volume of log data is huge.

The goal of the thesis was to reach situation awareness in computer systems and to reach it, it was chosen to implement monitoring system and log analytics platform. Monitoring system provides a high-level view and log analytics platform provides a low-level view into the system. All the basic functionality can be implemented with the open source versions of the chosen products. Both products support standard interfaces and protocols which make them highly extendable to support more devices and services. In addition of being extendable, the products and the introduced system architecture enables high scalability.

When comparing log analytics platforms cost versus commercial industry leaders starter packages cost, the difference was about half in favor of the open source alternative. That includes development costs, licensing costs and hardware resources. Development cost is a one-time investment and the next years will be a lot less expensive than the most popular on-premise commercial alternatives.

Research has been done in the field of secure logging to the cloud. For example, a study called ”Secure Logging as a Service—Delegating Log Management to the Cloud” addresses security and integrity issues during log generation, collection, transmission, storage and retrieval processes. [16] Log data is encrypted and anonymity of operations to and from the cloud is handled using an anonymizing network such as The Onion Router (Tor).

6.1 Future work

Nmap could be used fighting against firewall misconfiguration. Logstash has a codec for Nmap data, so it would be interesting to script Nmap to scan border firewalls and to report about changes. This data could be exported to Logstash, which would store the results to the Elasticsearch cluster. The history of scans could be browsed there and alerted if changed, like a last warning to administrators if they are sure to open a new port to the Internet.

The IDS/IPS service integrated into the firewall is used rather than separate well-known project like Snort[1], Suricata[2] or Bro[3]. Implementing support for any of those is a priority along with changing traffic logging from vendor-specific Syslog format to Netflow or IPFIX.

Finally, more visualizations and dashboards are needed in order to present the data in clearer and human-friendly ways but also new types of visualizations may expose new phenomenons. Especially interesting is graphing which is already a feature in X-Pack. Graphing reveals relationships between different documents, for example which machine communicates with which machines, and how much (amount of data or frequency). It could also easily reveal all the targets of a single attacker, for example when the attacker's IP address is known, where else did the same IP address appear at the time.

A survey from 2016 shows that there is a need for automation in security [17]. With this type of monitoring and logging system, the automation can be extended to cover more issues, which needs to be discovered and handled.

While traditional SIEM systems and anomaly detection systems have been seen as expensive and not having such a good value, an open source system can provide better value and flexibility. It is important to understand that no single technical solution is enough alone and even though automation is a good thing, it is currently not a good replacement for human understanding and insight.

The monitoring system and log analytics platform introduced in this thesis provides administrators a clear view into their computer system. Administrative work gets

[1]`https://www.snort.org`
[2]`https://suricata-ids.org`
[3]`https://www.bro.org`

easier when using these tools to automate everyday tasks and they provide triggering mechanisms to administrators when there is something which needs to be investigated. With the help of predictive monitoring and machine learning, upcoming outages and issues can be prevented and if something bad has already happened, the log analytics platform helps during incident response.

Support for the following systems was implemented during this thesis: Sonicwall firewalls and SRAs, HP Procurve switches and routers, Microsoft Windows Server audit framework (mostly object access, file share, Active Directory changes, policy changes), Microsoft Network Policy Server, Microsoft DHCP server, Linux audit framework, F-Secure Messaging Security Gateway, F-Secure Policy Manager and F-Secure Protection Service for Business.

REFERENCES

[1] M. Endsley, “SAGAT: a methodology for the measurement of situation awareness,” Northrop, Tech. Rep. Technical Report: NOR DOC 87-83, 1987.

[2] S. Jajodia, P. Liu, and V. Swarup, *Cyber Situational Awareness : Issues and Research*. Boston, MA: Springer, 2010;2009;, vol. 46.

[3] C. Lonvick, “The bsd syslog protocol,” Internet Requests for Comments, RFC Editor, RFC 3164, August 2001.

[4] D. New and M. Rose, “Reliable delivery for syslog,” Internet Requests for Comments, RFC Editor, RFC 3195, November 2001.

[5] F. Miao, Y. Ma, and J. Salowey, “Transport layer security (tls) transport mapping for syslog,” Internet Requests for Comments, RFC Editor, RFC 5425, March 2009.

[6] J. Kelsey, J. Callas, and A. Clemm, “Signed syslog messages,” Internet Requests for Comments, RFC Editor, RFC 5848, May 2010.

[7] C. Gates and C. Taylor, “Challenging the anomaly detection paradigm: A provocative discussion,” in *Proceedings of the 2006 Workshop on New Security Paradigms*, ser. NSPW ’06. New York, NY, USA: ACM, 2007, pp. 21–29. [Online]. Available: http://doi.acm.org/10.1145/1278940.1278945

[8] A. Tall, J. Wang, and D. Han, “Survey of data intensive computing technologies application to to security log data management,” in *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, ser. BDCAT ’16. New York, NY, USA: ACM, 2016, pp. 268–273. [Online]. Available: http://doi.acm.org/10.1145/3006299.3006336

[9] R. Marty, *Applied security visualization*. Addison-Wesley Upper Saddle River, 2009.

[10] I. Friedberg, F. Skopik, and R. Fiedler, “Cyber situational awareness through network anomaly detection: state of the art and new approaches,” *e & i Elektrotechnik und Informationstechnik*, vol. 132, no. 2, pp. 101–105, Mar 2015. [Online]. Available: https://doi.org/10.1007/s00502-015-0287-4

[11] W. Takase, T. Nakamura, Y. Watase, and T. Sasaki, “A solution for secure use of kibana and elasticsearch in multi-user environment,” *CoRR*, vol. abs/1706.10040, 2017. [Online]. Available: http://arxiv.org/abs/1706.10040

[12] S. Bagnasco, D. Berzano, A. Guarise, S. Lusso, M. Masera, and S. Vallero, “Monitoring of iaas and scientific applications on the cloud using the elasticsearch ecosystem,” *Journal of Physics: Conference Series*, vol. 608, no. 1, p. 012016, 2015. [Online]. Available: http://stacks.iop.org/1742-6596/608/i=1/a=012016

[13] D. Ferraiolo and R. Kuhn, “Role-based access control,” in *In 15th NIST-NCSC National Computer Security Conference*, 1992, pp. 554–563.

[14] R. Gerhards and C. Lonvick, “Transmission of syslog messages over tcp,” Internet Requests for Comments, RFC Editor, RFC 6587, April 2012.

[15] J. S. George Jones. (2014) Alternatives to signatures (alts). [Online]. Available: http://resources.sei.cmu.edu/asset_files/WhitePaper/2014_019_001_296152.pdf

[16] I. Ray, K. Belyaev, M. Strizhov, D. Mulamba, and M. Rajaram, “Secure logging as a service-delegating log management to the cloud,” *IEEE Systems Journal*, vol. 7, no. 2, pp. 323–334, 2013.

[17] AlgoSec. (2016, mar) The state of automation in security - an algosec survey. [Online]. Available: https://www.algosec.com/wp-content/uploads/2016/03/The-State-of-Automation-in-Security-Survey-Final.pdf

APPENDIX A. ELASTICSEARCH INDEX TEMPLATE FOR SONICWALL

```
{
  "template" : "*-sonicwall-*",
  "version" : 50013,
  "settings" : {
    "index.refresh_interval" : "5s",
    "number_of_shards": 1,
    "number_of_replicas": 1
  },
  "mappings" : {
    "sonicwall" : {
      "dynamic_templates" : [ {
        "message_field" : {
          "path_match" : "message",
          "match_mapping_type" : "string",
          "mapping" : {
            "type" : "text",
            "norms" : false
          }
        }
      }, {
        "string_fields" : {
          "match" : "*",
          "match_mapping_type" : "string",
          "mapping" : {
            "type" : "text", "norms" : false,
            "fields" : {
              "keyword" : { "type": "keyword", "ignore_above": 256 }
            }
          }
        }
      } ],
      "properties" : {
        "@timestamp": { "type": "date" },
        "@version": { "type": "keyword" },
        "avgThroughput": { "type": "long" },
        "bytesIn": { "type": "long" },
        "bytesOut": { "type": "long" },
        "bytesTotal": { "type": "long" },
        "domain": { "type": "keyword" },
        "dst": { "type": "text" },
        "dst_if": { "type": "keyword" },
        "dst_ip": { "type": "ip" },
        "dst_port": { "type": "long" },
        "dst_vlan": { "type": "long" },
        "dst_geoip"  : {
          "dynamic": true,
          "properties" : {
            "ip": { "type": "ip" },
            "location" : { "type" : "geo_point" },
            "latitude" : { "type" : "half_float" },
            "longitude" : { "type" : "half_float" }
          }
        },
        "duration": { "type": "long" },
        "event": {
          "type": "text",
          "fields": {
            "raw": {
              "type": "keyword"
            }
          }
        },
        "event_id": { "type": "long" },
        "fw": { "type": "ip" },
        "host": { "type": "ip" },
        "maxThroughput": { "type": "long" },
        "message": { "type": "text" },
        "msg": { "type": "text" },
        "portal": { "type": "keyword" },
        "pri": { "type": "long" },
        "proto": {
          "type": "text",
          "fields": {
            "raw": {
              "type": "keyword"
            }
          }
        },
        "rcvd": { "type": "long" },
        "rule": { "type": "keyword" },
        "sent": { "type": "long" },
        "src": { "type": "text" },
        "src_if": { "type": "keyword" },
        "src_ip": { "type": "ip" },
        "src_port": { "type": "long" },
        "src_vlan": { "type": "long" },
        "src_geoip"  : {
          "dynamic": true,
          "properties" : {
            "ip": { "type": "ip" },
            "location" : { "type" : "geo_point" },
            "latitude" : { "type" : "half_float" },
            "longitude" : { "type" : "half_float" }
          }
        },
        "usr": { "type": "keyword" },
        "vpnpolicy": { "type": "text" },
        "application": { "type": "keyword" },
        "customer": { "type": "keyword" },
        "valvonta": { "type": "keyword" }
      }
    }
  }
}
```

Configuration 1 *Example of Elasticsearch index template for Sonicwall firewalls and Secure Remote Appliances.*

APPENDIX B. LOGSTASH FILTER FOR SONICWALL

```
filter {
  if [type] == "sonicwall" {
    # Parse fields using key-value filter
    kv {
      include_keys => [ "avgThroughput", "bytesIn", "bytesOut", "
         bytesTotal", "domain", "dst", "duration", "fw", "m", "
         maxThroughput", "msg", "pri", "proto", "portal", "rcvd", "
         rule", "sent", "src", "time", "usr", "vpnpolicy" ]
      add_tag => [ "sonicwall" ]
      add_field => { "monitoring" => "Site_Firewall" }
    }

    # Replace generated timestamp with real timestamp from Sonicwall
       log message
    if "UTC" in [time] {
      date {
        match => [ "time", "yyyy-MM-dd HH:mm:ss 'UTC'" ]
        timezone => "UTC"
        remove_field => [ "time" ]
      }
    }
    else {
      date {
        match => [ "time", "yyyy-MM-dd HH:mm:ss" ]
        timezone => "Europe/Helsinki"
        remove_field => [ "time" ]
      }
    }

    # Rename cryptic "m" field to event_id
    mutate {
      rename => { "m" => "event_id" }
    }

    # Translate event_id into human-readable messages
    translate {
      field => "event_id"
      destination => "event"
      dictionary_path => "/etc/logstash/translate/sonicwall-ids-event.
         yml"
    }

    # If source field exists, parse it.
    if [src] {
      if ":" not in [src] {
        # If there is no separator in src-field, it is IP address
        mutate {
          copy => { "src" => "src_ip" }
        }
      }
      else if [src] =~ "^::(.*)" {
        # Split source into an interface
        mutate {
          split => { "src" => ":" }
          add_field => {
            "src_if" => "%{[src][2]}"
          }
        }
      }
      else if [src] =~ "(.*)::$" {
        # Split source into IP address
        mutate {
          split => { "src" => ":" }
          add_field => {
            "src_ip" => "%{[src][0]}"
          }
        }
      }
      else if [src] =~ "^:(.*):(.*)" {
        # Split source into port and interface
        mutate {
          split => { "src" => ":" }
          add_field => {
            "src_port" => "%{[src][1]}"
            "src_if" => "%{[src][2]}"
          }
        }
      }
      else if [src] =~ "(.*)::(.*)" {
        # Split source into IP address and interface
        mutate {
          split => { "src" => ":" }
          add_field => {
            "src_ip" => "%{[src][0]}"
            "src_if" => "%{[src][2]}"
          }
        }
      }
      else if [src] =~ "(.*):(.*):(.*)" {
        # Split source into IP address, port and interface
        mutate {
          split => { "src" => ":" }
          add_field => {
            "src_ip" => "%{[src][0]}"
            "src_port" => "%{[src][1]}"
            "src_if" => "%{[src][2]}"
          }
        }
      }
      else if [src] =~ "^:(.*)" {
         # Split source into port
         mutate {
          split => { "src" => ":" }
          add_field => {
            "src_port" => "%{[src][1]}"
          }
        }
      }
      else if [src] =~ "(.*):(.*)" {
         # Split source into IP address and port
         mutate {
          split => { "src" => ":" }
          add_field => {
            "src_ip" => "%{[src][0]}"
            "src_port" => "%{[src][1]}"
          }
        }
      }
      else {
        mutate {
          add_tag => [ "src_parse_failure" ]
        }
      }
    }

    # If destination field exists, parse it.
    if [dst] {
      if ":" not in [dst] {
        # If there is no separator in src-field, it is IP address
        mutate {
          copy => { "dst" => "dst_ip" }
        }
      }
      else if [dst] =~ "^::(.*)" {
        # Split destination into an interface
        mutate {
          split => { "dst" => ":" }
          add_field => {
            "dst_if" => "%{[dst][2]}"
          }
        }
      }
      else if [dst] =~ "(.*)::$" {
        # Split destination into IP address
        mutate {
          split => { "dst" => ":" }
          add_field => {
            "dst_ip" => "%{[dst][0]}"
          }
        }
      }
      else if [dst] =~ "^:(.*):(.*)" {
        # Split destination into port and interface
        mutate {
          split => { "dst" => ":" }
          add_field => {
            "dst_port" => "%{[dst][1]}"
            "dst_if" => "%{[dst][2]}"
          }
        }
      }
      else if [dst] =~ "(.*)::(.*)" {
        # Split destination into IP address and interface
        mutate {
          split => { "dst" => ":" }
          add_field => {
            "dst_ip" => "%{[dst][0]}"
            "dst_if" => "%{[dst][2]}"
          }
        }
      }
      else if [dst] =~ "(.*):(.*):(.*)" {
        # Split destination into IP address, port and interface
        mutate {
          split => { "dst" => ":" }
          add_field => {
            "dst_ip" => "%{[dst][0]}"
            "dst_port" => "%{[dst][1]}"
            "dst_if" => "%{[dst][2]}"
          }
        }
      }
      else if [dst] =~ "^:(.*)" {
        # Split destination port
        mutate {
          split => { "dst" => ":" }
          add_field => {
            "dst_port" => "%{[dst][1]}"
          }
        }
      }
      else if [dst] =~ "(.*):(.*)" {
        # Split destination into IP address and port
        mutate {
          split => { "dst" => ":" }
          add_field => {
            "dst_ip" => "%{[dst][0]}"
            "dst_port" => "%{[dst][1]}"
          }
        }
      }
      else {
        mutate {
          add_tag => [ "dst_parse_failure" ]
        }
      }
    }

    # If source interface fields exists, split it into interface and
       VLAN
    if [src_if] {
      if "-" in [src_if] {
        mutate {
          split => { "src_if" => "-" }
          add_field => { "src_vlan" => "%{[src_if][1]}" }
        }
      }
    }

    # If destination interface field exists, split it into interface
       and VLAN
    if [dst_if] {
      if "-" in [dst_if] {
        mutate {
          split => { "dst_if" => "-" }
          add_field => { "dst_vlan" => "%{[dst_if][1]}" }
        }
      }
    }

    # Remove "V" from VLAN ID and replace array with interface value
    if [src_vlan] {
      mutate {
        gsub => [ "src_vlan", "V", "" ]
        replace => { "src_if" => "%{[src_if][0]}" }
      }
    }

    # Remove "V" from VLAN ID and replace array with interface value
    if [dst_vlan] {
      mutate {
        gsub => [ "dst_vlan", "V", "" ]
        replace => { "dst_if" => "%{[dst_if][0]}" }
      }
    }

    # Remove src and dst arrays
    if "dst_parse_failure" not in [tags] or "src_parse_failure" not in
       [tags] {
      mutate {
       remove_field => [ "src", "dst" ]
      }
    }
  }
}
```

***Configuration** 2 Example of Logstash filter for Sonicwall firewalls and Secure Remote Appliances.*