Expertise
and insight
for the future

Topi Talikka

Implementing Linux support and monitoring in a workplace

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

9 April 2019

Author Title Number of Pages Date	Topi Talikka Implementing Linux support and monitoring in a workplace 26 pages 9 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Internet of Things and Cloud Computing
Instructors	Tapio Wikström, Senior Lecturer

The goal of the thesis was to improve the condition of Linux support, monitoring and automation in a workplace. The endpoint enrollment was investigated, alongside how software distribution could be made more straightforward.

After preliminary planning, the automation aspect of the project was agreed to be handled by Ansible. The IT organization would provide Ansible playbooks, which the end users could run themselves, depending on the configuration they want installed. The playbooks could provide easy-to-use solutions for the end users, and could automate processes such as installing printer queues, installing developer tools (version control, integrated development environments, text editors), and installing analytics tools. The playbooks are open-source inside the company, so anyone can make pull requests on the code if they find it necessary to improve it.

Monitoring Linux endpoints is necessary to the company after a preliminary ISO27001 certification. This was achieved by a combination of different software solutions, which all depended on the endpoints running Osquery.

The project was successful, and the same platforms and tools were later taken into use for Windows and Mac devices as well. The project generated a lot of value in the form of information for the company.

Keywords	Linux, Ansible, Osquery, Kolide Fleet, Graylog

Tekijä Otsikko Sivumäärä Aika	Topi Talikka Linux-tuen kehitys ja monitorointi yrityksessä 26 sivua 9.4.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Asioiden Internet ja Pilvipalvelut
Ohjaajat	Tapio Wikström, Lehtori

Insinöörityön tavoitteena oli parantaa yrityksen Linux-tukea, monitorointia sekä automatisointia.

Alustavan suunnittelman mukaan kaikki työhön liittyvä automatisointi päätettiin toteuttaa Ansible-ohjelmistoa käyttämällä. Yrityksen IT-organisaatio tarjoaa Ansiblen käyttämät mallit eli pelikirjat (playbook), jotka käyttäjät voivat itse suorittaa Ansiblella. Eri pelikirjoja käyttämällä käyttäjät saavat erilaisia konfiguraatioratkaisuja päätelaitteillensa.
Pelikirjat tarjoavat käyttäjille helppokäyttöisiä automaatioratkaisuja useaan eri tarpeeseen. Näitä ovat esimerkiksi tulostusjonojen sekä -ajureiden asentaminen ja kehitystyökalujen automatisoitu asentaminen. Näiden lisäksi pelikirjoilla asennetaan analytiikka- ja monitorointityökalu Osquery.Pelikirjat ovat saatavilla avoimena lähdekoodina yrityksen sisällä, joten yrityksen työntekijät voivat tarkastella niitä tai ehdottaa niihin parannuksia.

Linux-päätelaitteiden monitoroinnista tehtiin pakollista, kun yrityksessä tehtiin ISO27001-auditointi. Monitorointi toteutettiin, ja se pohjautui erityisesti Osquery-ohjelmistoon.

Projekti onnistui hyvin, ja se loi IT-organisaatiolle paljon arvoa informaation muodossa. Projektissa tutkittuja ja käyttöönotettuja järjestelmiä sekä työkaluja käytettiin niiden toimivuuden toteamisen jälkeen myös Windows- ja Mac-järjestelmillä.

Avainsanat	Linux, Ansible, Osquery, Kolide Fleet, Graylog

Contents

List of Abbreviations

1 Introduction 1

2 History of Linux 2

3 Planning the structure of the project 3

3.1.1 Automating Linux deployments 3
3.1.2 Monitoring Linux endpoints 4

4 Implementation 4

4.1 Kolide Fleet 4
4.1.1 Taking Fleet into use in Amazon Web Services 5
4.1.2 Communicating with Graylog 6
4.2 Ansible 8
4.2.1 Using Ansible 9
4.2.2 Playbooks 10
4.3 Graylog 11
4.3.1 Queries 12
4.3.2 Dashboards 13
4.4 Utilizing the logging with the ISO27001 certification 16
4.4.1 VPN authorization 16
4.4.2 Monitoring for security risks 19

5 Summary 25

References

List of Abbreviations

AD	Active Directory. A directory service developed by Microsoft for Windows domain networks.
API	Application Programming Interface. A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service. [11]
APT	Advanced Package Tool. User interface that handles the installation and removal of software on Debian-based operating systems.
AWS	Amazon Web Services. A subsidiary of Amazon.com which provides on-demand cloud computing platforms.
CPU	Central Processing Unit. The circuitry in the computer that carries out the instructions provided by a computer program.
CUPS	Common UNIX Printing System. A modular printing system which allows the endpoint to work as a printing server and process print jobs.
DPKG	Debian Package. A low-level tool used to install and remove .deb packages.
EC2	Amazon Elastic Compute Cloud. A web service which provides compute capacity in cloud.
GCC	GNU Compiler Collection. A compiler system supporting various programming languages.
GNU	GNU's Not Unix! An operating system and a collection of free computer software for Unix-like systems.
HTTP(S)	Hypertext Transfer Protocol (Secure). A protocol for browsers and WWW-servers to communicate with in hypermedia information systems.
JSON	JavaScript Object Notation. A human-readable data format.

Kernel	Kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system.
PPD	PostScript Printer Description. A feature set for PostScript printers.
RAM	Random Access Memory. The computer data storage for data currently being used.
Regex	Regular Expression. A syntax for defining search patterns in text.
RPM	RPM Package Manager. Open-source package management system.
S3	Amazon Simple Storage Service. An object storage service provided by Amazon with a web service interface.
SSH	Secure Shell. A protocol for secure operating of network services over unsecured networks.
VPN	Virtual Private Network. A technology that extends a private network, enabling the users to secure send and receive data over public networks.
YAML	YAML Ain't Markup Language. A minimal human-readable data serialization language format.
XML	Extensible Markup Language. A markup language.

1 Introduction

As of 2018, Linux is still a minority in desktop operating systems. Its worldwide usage is, depending on the survey, around or slightly less than 2 percent. However, according to a survey that Stack Overflow conducted on 2018, the usage of Linux as a desktop operating system skyrockets up to 23 percent when asking developers what operating systems they use for work [1]. This means that Linux cannot be disregarded as a major operating system, especially when working in a developer-driven company.

The goal of this thesis was to improve the situation of the Linux-environment in a workplace. This included automating tasks, installations and such, and improving the support provided by the IT department. When the project started, the company had around 20-30 Linux users (between 5-10 percent of all employees) even though the IT department provided next to no support.

A large part of the project quickly started revolving around an ongoing ISO27001 certification, which started around the same time as the project did. The certification required meeting various specifications regarding the information security of the company, an aspect overlooked or neglected on the Linux endpoints. The certification demanded that various aspects of the endpoints should be monitored. This was achieved by the combination of Osquery, Kolide Fleet and Graylog.

2 History of Linux

Linux is an umbrella term for a large family of free and possibly open-source operating systems running on top of a Linux kernel. The kernel was developed in the early nineties by a Finnish undergraduate Linus Torvalds, after whom the operating system family is named.

The roots of Linux are much farther in the past, though. Linux was built as an open-source alternative to Unix, which was the result of a development project started in Bell Laboratories' Computer Sciences Research Center. It was an iteration of a project called Multics, which had it funding stopped in 1969. A group of researchers decided to continue working on the project anyway. The project was rewritten in C in 1972-1973, which resulted in Unix going portable. It could be moved between devices and could run on different hardware setups, which was unique to operating systems of that time. [2]

In 1991, Linus Torvalds was frustrated with the licensure of MINIX, an Unix-like operating system developed by Andrew S. Tanenbaum in 1987. He started working on his own operating system, which initially resembled MINIX. The project eventually grew away from MINIX, and the 1.0 version of Linux kernel was released in 1994 by Torvalds and a group of developers. The kernel would later be used by hundreds of Linux distributions, including the most popular ones like Debian, Red Hat, Fedora, CentOS, Linux Mint and Kali Linux, to name a few. Linux is also the base of the Android operating system, and is used on virtually all embedded systems. [2]

3 Planning the structure of the project

3.1.1 Automating Linux deployments

The project started in late September 2018 when the company I worked for decided that it was time to improve the Linux support it was providing. Until then, only Windows and Mac were the supported operating systems in the house although the users had the option to use Linux. The Internal IT did not have resources to support the Linux environment, even though more and more users were starting to adopt it. At this point, there were approximately 20-30 Linux users in the company, with around a total of 400 users.

The project started by investigating what the users wanted. I started drawing guidelines with my manager, who is also the supervisor of this project, on what the Internal IT should provide support on and what is beyond the departments control. Several different automation solutions were discussed, since it would have been good to automate the deployment procedures as much as possible. Ansible, which is an open source system management software, was selected to handle the automation. It features an agentless architecture, which means that the managed devices do not need to run any additional daemons in the background, thus no resources are consumed when idle. [3] Although presented with the possibility of remotely managing the inventories of all Linux machines, we decided against it, and rather left the end users in control of their own systems. There were only a couple of mandatory packages that had to be installed on all systems and and a decision was made to just use internal policies to have the end users install the required software. Softwares such as Puppet and Spacewalk were also considered, but ultimately Ansible proved to be the best considering the requirements of the project.

I created a chatroom in Google Hangouts Chat, the internal messaging software of the company, for all Linux users in the company. The discussion started off by asking the users what different Linux distributions and hardware they were using, and if there were any software installations or similar they would like to have automated. We did not get many suggestions on the software packages but got the first insights on what sort of

systems the users were running. The distributions included Debian, Fedora, Mint, Ubuntu and Kubuntu, to name the most prominent. This meant that the playbooks the Internal IT would provide would have to include support for both .rpm and .deb packages and take into consideration the differences of RPM and DPKG.

3.1.2 Monitoring Linux endpoints

Around the same time as the project started, the company also underwent an ISO27001 preliminary certification. The certification demanded that the company should get better visibility into the Linux clients and enforce some aspects, for example, full disk encryption. This provided a good starting point to automate and enforce the end users to take a monitoring solution into use.

The monitoring solution that was decided to take into use was Kolide Fleet. It is an open-source Osquery manager, provided by a company called Kolide. It serves as both a front-end as well as a back-end for running Osquery queries on a target system. [4] Osquery is a multi-platform open-source operating system instrumentation framework, originally written by Facebook for their own use. It allows the user to query systems with SQL-based queries to explore operating system data. [5] The data from the queries executed by Fleet using Osquery is then propagated into Graylog log management system, which was already set up and used before this project. Graylog is then used to store and analyze the log information flowing from Fleet.

4 Implementation

4.1 Kolide Fleet

Fleet binary provided by Kolide includes the Fleet web interface, the Fleet application API endpoints and the Osquery TLS server API endpoints. These are served through a built-in HTTP server provided by the installation binary, so taking Fleet into use is fairly straightforward. The Fleet can be installed straight from the binary file, or by using Docker. The decision was made to install the Fleet straight from the binary file.

4.1.1 Taking Fleet into use in Amazon Web Services

Internal IT was already using AWS to some extent, and wanted to move as much as possible from the on-premises network into the Amazon cloud. Thus, installing Fleet into an AWS EC2 instance was seen as the best approach.

An EC2 instance model was chosen, in this case a t2.small with a single virtual CPU and 2 gigabytes of RAM. This instance was spawned into the eu-central region in AWS. Ubuntu Server was selected as the underlying operating system, running the 18.04 Bionic version. The instance was given a public IP address, and security groups were designed to allow SSH access to the instance from the internal company network and HTTPS access from anywhere.

The infrastructure was ready for Fleet installation at this point. The binaries were downloaded and unzipped with the commands listed in Listing 1.

```
curl -O https://dl.kolide.co/bin/fleet_latest.zip
unzip fleet_latest.zip 'linux/*' -d fleet
```

Listing 1. An example of downloading and unzipping Fleet.

After the binary was downloaded and extracted, the binary could be tested by executing the binary with the command *.fleet_linux_amd64 --help*.

The Fleet web server requires a MySQL database as its primary database and a Redis instance to ingest and queue the results of distributed queries. [7] These had to be setup at AWS as well.

A Redis-node with the name *kolide-internal* was setup in the eu-central-1b availability zone in AWS ElastiCache, and routed into the internal production network. The network setup was done previously when the AWS environment was originally set up.

The MySQL instance was set up in AWS Relational Database Service, or RDS for short. The database of 20 gigabytes was created using MariaDB, an open-source fork of MySQL.

After the required dependencies were set up, the Kolide configuration file was created with the required parameters pointed to the AWS, as shown in Picture 1 below.

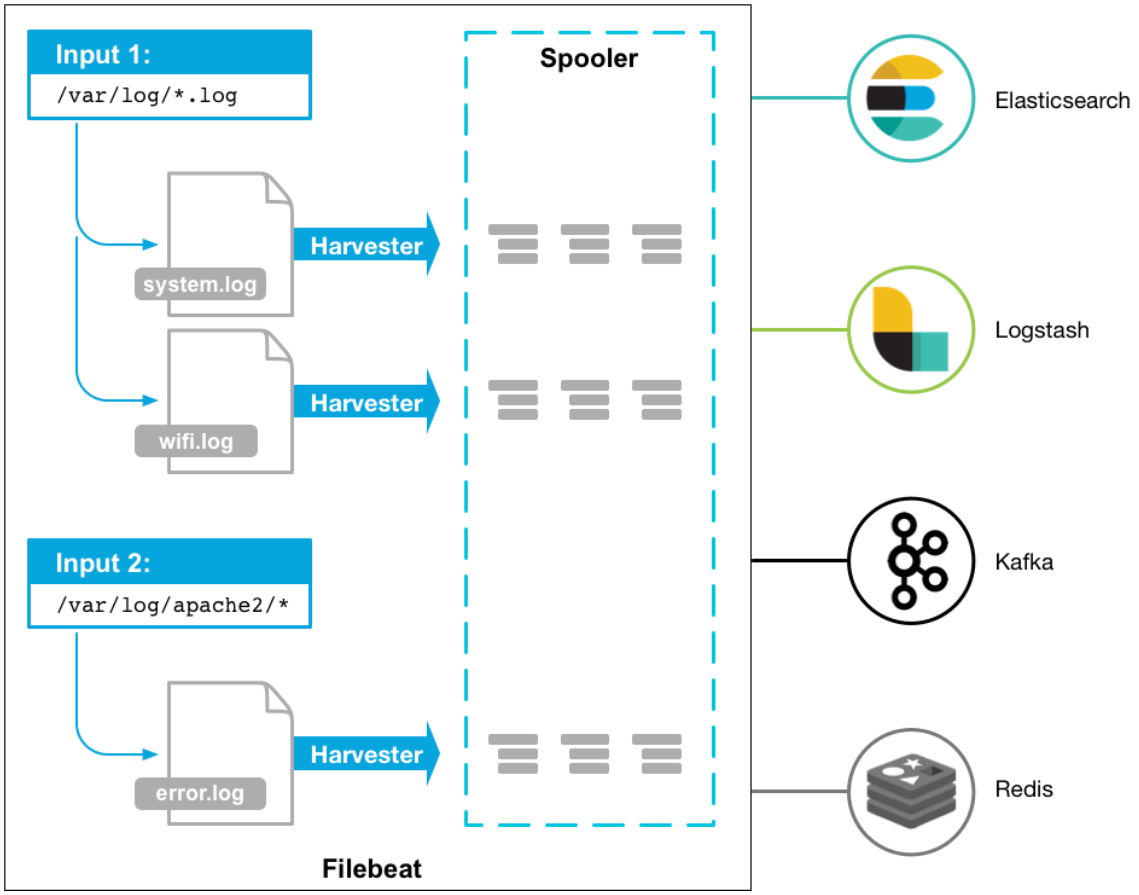
```
[redacted]@[redacted]:~$ cat /etc/kolide.conf
mysql:
  address: [redacted].eu-central-1.rds.amazonaws.com:3306
  database: [redacted]
  username: [redacted]
  password: [redacted]
redis:
  address: [redacted]cache.amazonaws.com:6379
server:
  cert: /installs/ssl/server.crt
  key: /installs/ssl/server.key
  tls: true
  address: 0.0.0.0:443
logging:
  debug: false
  json: true
auth:
  jwt_key: [redacted]
osquery:
  status_log_file: /var/log/osquery/status.log
  result_log_file: /var/log/osquery/result.log
  label_query_update_interval: 1h
  enable_log_rotation: true
```

Picture 1. Kolide configuration file

After Fleet had been configured, endpoints could be enrolled into it and the logging could be started. The first endpoints were added manually, by downloading the launcher binary and providing it with the hostname of the Fleet backend, a root directory for the local database and the enroll secret provided by the backend [9]. Only after the system had been tested and deemed useful, a configured binary was built and distributed.

4.1.2 Communicating with Graylog

After the first test devices started communicating properly with the Fleet backend, the logs had to be sent to Graylog for further analysis and monitoring. This was done using Filebeat, which is a shipper for forwarding log data. It monitors the log files and collects log events, and ultimately sends the aggregated events to Elasticsearch, which is used in the Graylog backend. [10] A reference image can be seen in Picture 2.



Picture 2. Filebeat reference [10]

In Picture 2, Filebeat looks at the log files residing in */var/log/*.log* and */var/log/apache2/*.* In this use case, the only log file that had to be forwarded was the log gathered by Fleet, which is located by default in the file */var/log/osquery/result.log.* Filebeat was then configured to forward the logs to Graylog, which was already running internally.

After running for a couple of weeks without problems, Fleet suddenly stopped sending logs to Graylog. The culprit was quickly discovered when an SSH connection was opened with the server and the shell completion would not work. This was because the Fleet log files had completely filled the hard disk of the server instance, prohibiting it from doing anything. The log files were flushed and a log rotation was implemented using a Linux utility called *logrotate*. The utility compresses the old logs and creates new log files to be used instead. Logs old enough get deleted.

AWS also offers a ready-made monitoring solution for Linux instances, monitoring the disk and memory metrics on an EC2 instance such as ours. The metrics are sent to

Cloudwatch to be monitored and possibly to be used as alarm definitions. This was also taken into use to make sure that the disk space would not completely fill up again. The instructions for taking the solution into use can be found here: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Install-CloudWatch-Agent.html

4.2 Ansible

Ansible was used in the project for installations and other automation tasks. It is a simple, agentless, open-source automation tool. It is designed to be used over SSH on multiple endpoints by a system administrator, but a different approach was chosen for the project. In the use case of this project, the end users were responsible for running the "playbooks" (an Ansible term for collections of different tasks), which would be built by Internal IT to provide tool and driver installations and configuration changes, among other tasks.

One of the reasons of using Ansible, or any automation frameworks, was the compatibility between various Linux distributions. Ansible playbooks were easy to write and read, and worked, up to a certain limit, similarly on all distributions, independent from the shell of the operating system.

Ansible is also much more simple than a shell script. The playbooks are written in YAML, which is easier for humans to read than XML or JSON. [8] In the Listing 2 below, the process of checking and installing two APT packages is shown.

```
---
  - name: Install smbclient and cups
    become: true
    become_method: sudo
    apt:
     name: "{{packages}}"
     state: present
    vars:
     packages:
      - smbclient
      - cups
```

Listing 2. An example of installing two APT packages

4.2.1 Using Ansible

The structure of the Ansible directory of the project is as follows: The master playbook of the project is called *site.yml* and it resides in the root directory, alongside with the *hosts* file which defines the hosts on which to run the Ansible playbooks, *ansible.cfg* configuration file which defines the *hosts* file as the inventory of the project and *RE-ADME.md* which provides instructions and information for the projects Github repository. Listing 3 provides an overview of the master playbook *site.yml*.

```
---
# file: site.yml

  - name: find out which OS are we running
    hosts: all
    tasks:
      - name: Classify hosts depending on their OS distribution
        group_by:
          key: os_{{ ansible_facts['distribution']|replace(' ','') }}
      - debug:
          msg: "key: os_{{ ansible_facts['distribution']|replace(' ','') }}"

  - hosts: "{{deblist}}"
    gather_facts: False
    roles:
      - role: debian
    vars:
     deblist:
      os_Ubuntu
      os_Debian
      os_LinuxMint

  - hosts: "{{rpmlist}}"
    gather_facts: False
    roles:
      - role: rpm
    vars:
     rpmlist:
      os_Fedora
      os_CentOS
      os_RHEL
      os_SUSE
```

Listing 3. Site.yml file

Site.yml playbook file sets the roles for Ansible to use. The first task that is executed examines the operating system distribution that it is currently running on. It does this by checking the ”facts” that Ansible has collected, specifically the distribution fact. It then uses variables to determine which role the machine should use to execute any additional playbooks.

Under the root directory also lives a *roles*-directory, which includes two additional directories, *debian/tasks* and *rpm/tasks*. Different Linux distributions require different components on certain applications, and require slightly different task sequences as a consequence.

In these directories are the actual task sequence playbooks. Both of the directories include a *main.yml* file, which imports the task sequences and assigns tags to them. This allows the users to skip installations that they do not want or require. The playbooks for different applications or task sequences are in separate files, allowing the segregation of the tasks, providing clarity for the project.

4.2.2 Playbooks

The first playbook created was called *symantec.yml.* It was a playbook that downloaded an archived file from an AWS S3 bucket, unarchived it, created some directories, installed dependencies, modified some permissions and finally ran the unarchived installer. It was created to streamline the installation of Symantec Endpoint Protection, an endpoint protection software that was ultimately scrapped due to its poor Linux support.

As Internal IT wanted to install Kolide Fleet on all endpoints to get visibility to them, a playbook for installing it was created. This playbook simply downloaded the customized software package built for the Kolide endpoint software and installed it.

The largest playbook created was used to install secure print queues to the end users machines. This included installing the Canon drivers, GCC packages and libraries and Canon CUPS PPD files. Printer queues also had to be configured. Canon provided a shell script for installing and configuring the forementioned, which had to be dissected and rewritten for Ansible use.

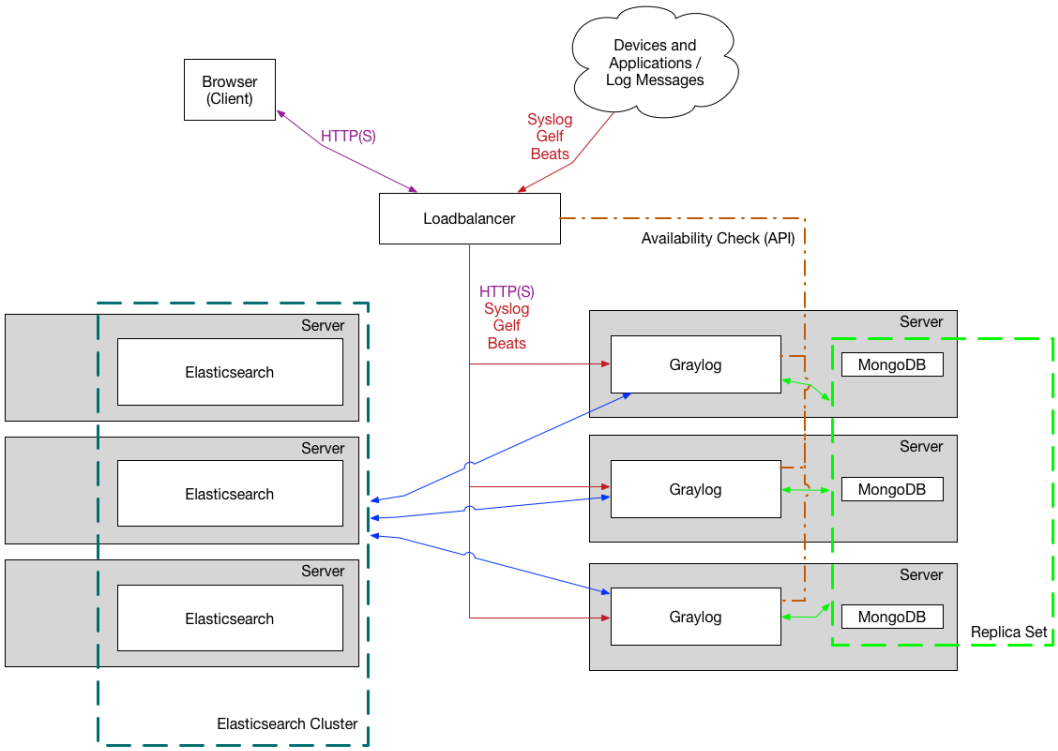
An internal Github repository was created for the Ansible playbooks and configuration files. This allowed the end users to easily observe the source code and the changes on the playbooks. The repository also allowed the end users to raise issues with the playbooks or create change requests for them. This greatly improved the structure of the playbooks, as more experienced Ansible users could suggest improvements on them.

Error rates and issues also lowered considerably as the users could fix the playbooks themselves if they encountered any errors in them.

4.3 Graylog

Graylog handles the log management of the project. It is an open source solution that the company was actively using, and forwarding logs from Kolide Fleet to it was the most sensible decision.

Graylog Server is composed of at least one instance of Graylog Server, MongoDB and Elasticsearch. They all serve different purposes, and at least one instance of everything must be included. The Graylog Server nodes are the workers that receive and process the incoming messages, while also serving the user interface. The processed messages are passed on to Elasticsearch, which stores them. MongoDB is used to store the metadata and the configuration data. [6] Picture 3 shows a complex multi-node architecture of a Graylog system.

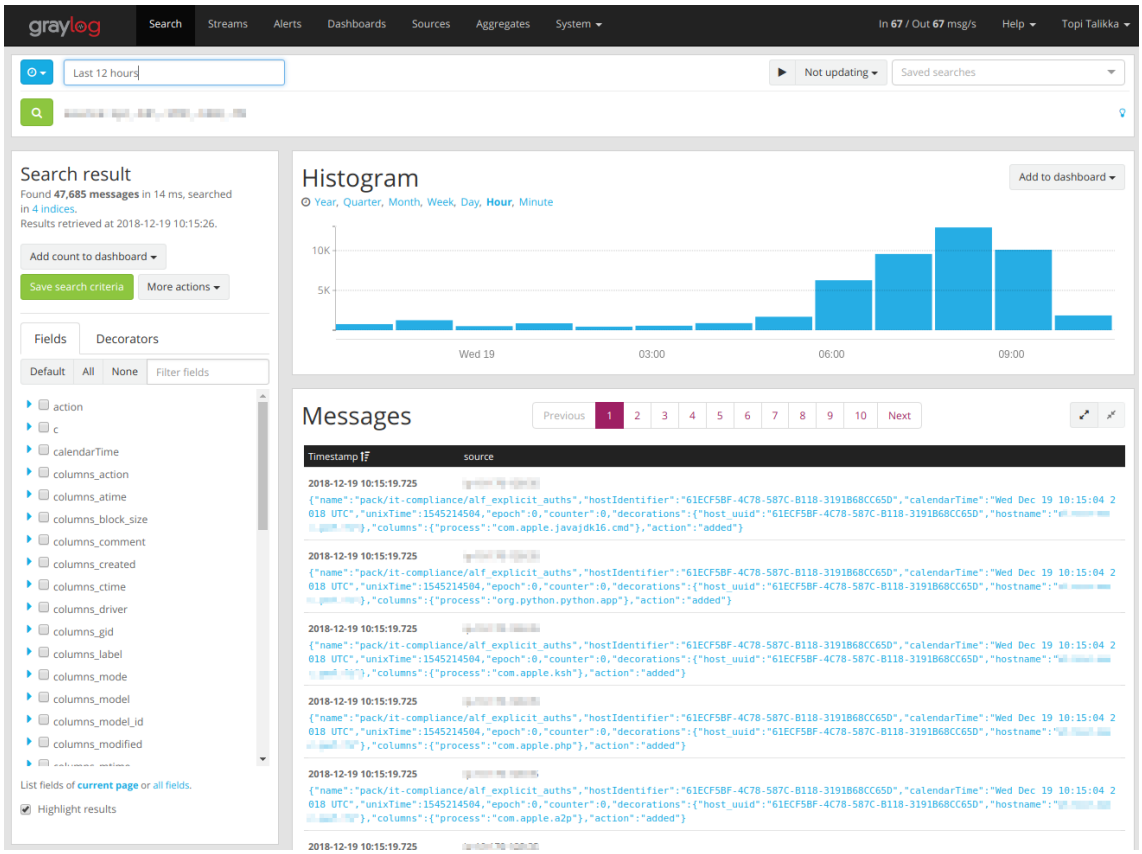


Picture 3. Complex multi-node architecture of a Graylog system. [6]

The company currently utilizes a single node running virtualized in VMware vSphere, using 12 vCPUs and 12 gigabytes of RAM. The system logs averagely around seven million messages per day, of which around 100 000 are from the monitoring system discussed in this thesis.

4.3.1 Queries

Messages and logs can be queried in Graylog. Kolide Fleet sends the messages in JSON format, which can be dissected in Graylog and used as search operators. Graylog also saves the source of the message, which allows the users to search for all logs from a single or multiple sources. This is demonstrated in Picture 4.



Picture 4. Example of a Graylog query.

The search query in the Picture 4 shows all the messages sent from Fleet in the last twelve hours. The Messages-field shows the timestamp of when the message was logged into Graylog, the source and the dissected message, also including the original JSON message.

The query function is especially useful when events from a pack defined in Kolide or from a single user have to be investigated.

4.3.2 Dashboards

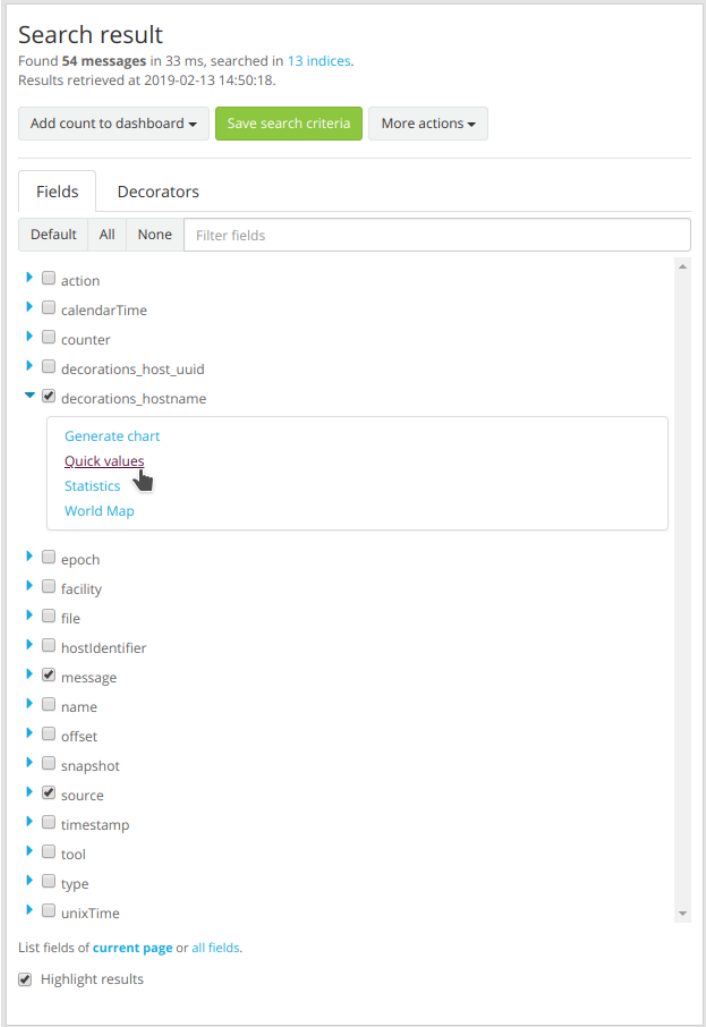
Graylog also allows the users to create dashboards from the query data. This is especially useful for matters that the IT has to check every now and then, for example, the disk encryption status of endpoints enrolled into Kolide. The query in Listing 4 provides an example of such a query.

```
source: hostname AND message: "*pack\/it\-compliance\/disk_encryption*" AND
NOT message: "Executing scheduled query pack*" AND NOT message: "en-
crypted\":\"1"
```

Listing 4. An example of querying for the disk encryption status

The example provided in sets the source of the query as the Kolide server, selects the correct pack that Kolide logs, and creates some rules that trim out the unwanted results (notifications of a scheduled query, encrypted volumes). The query only returns the results, in this particular case the endpoints that do not have any encrypted volumes.

The query itself only returns a long list of JSON-messages, which are hard to look at if the user quickly wants the information of machines that do not have any encrypted partitions. Creating a list of hostnames matching the criteria makes the task much easier, as seen in Pictures 5 and 6.



Picture 5. Creating a quick value list of query results

Linux and Mac machines without encryption

Value	%	Count
Top 1000 values		
[redacted]	24.07%	13
[redacted]	18.52%	10
[redacted]	16.67%	9
[redacted]	14.81%	8
[redacted]	11.11%	6
[redacted]	11.11%	6
[redacted]	1.85%	1
[redacted]	1.85%	1

Picture 6. An example of a quick value list

Picture 6 shows the selection of the endpoint hostnames as the criteria of the quick value listing, which results in the list presented in the picture. The count does not show the actual amount of unencrypted volumes on the endpoint, but rather the amount of times the query has automatically been executed on the machine. The query was set to show the messages from the last 30 days.

4.4 Utilizing the logging with the ISO27001 certification

4.4.1 VPN authorization

One of the reasons that the centralized management and the logging/monitoring of the endpoints was the ISO27001 information security certification. The ISO auditors demanded the company to monitor and enforce certain aspects. One condition that the certification and the auditors required was to only allow the company owned devices to connect to the internal VPN. For Linux machines, this was accomplished by a combination of Kolide Fleet, Graylog and its API, and Python.

The effort on the VPN had to be coordinated with the Systems and Operations (Sysops) department of the company, as they were responsible for the VPN connections in and out of the company network. After some investigating and negotiations, a decision was made to use a hashing function built in to the VPN software on a certain serial file located in all Linux systems, and compare that with a list of hashes of the same file, gathered from all enrolled Linux machines.

For this, a query had to be created first into Kolide Fleet, to be executed automatically on all Linux computers in the fleet. Osquery has a built-in hash function, which allows it to calculate multiple hashes from the files in the system it is running on. Since the VPN software calculated the file hashes as MD5 hashes, this allowed us to use the same function on Osquery, as MD5 was natively supported on it. The query is demonstrated in Listing 5.

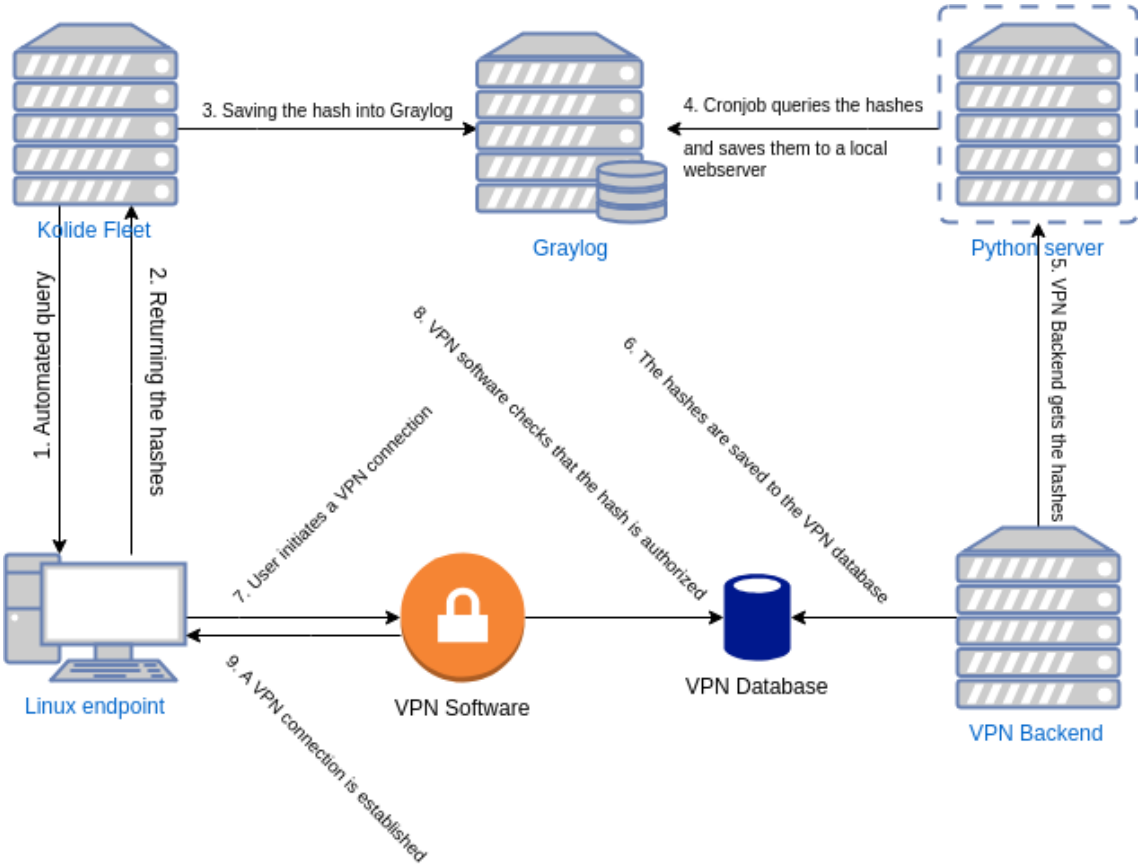
```
SELECT hostname, md5 FROM hash WHERE path = "/sys/class/dmi/id/board_serial"
```

Listing 5. An Osquery query that returns the hostname and the MD5 hash of a file

The query returns the hostname and the calculated MD5 hash of the board serial file. This query was set up in Kolide Fleet to run on all Linux machines every 3600 seconds, or one hour. Only the differential would be logged into Graylog, so the consecutive queries would not fill Graylog with the same MD5 hash as before, only the changed values would be logged again.

After setting up the query pack in Kolide Fleet, the endpoints will start to report the hashes of the board_serial file back into Fleet, which are then propagated to Graylog. The information has to be extracted from there to the VPN software.

For this task I set up virtual machine, running in the same local network as our Graylog server. On the virtual machine, I set up a Python script utilizing the Graylog API, making API calls to query for machines that have a board_serial hash reported into Graylog. The hostnames and the corresponding hashes were then listed into a simple text file, which was written into the public HTML directory of the virtual machine. This allowed the VPN software to read the file, and add the values to the database that it was using to authorize the clients. The process is shown visually in Picture 7.



Picture 7. A flowchart of the VPN authorization process

The code running on the Python server is shown in Listing 6 below.

```
import requests
import json
import datetime
url = 'http://255.255.255.255:9000/api/search/universal/rela-
tive?query=source%3Aip%5C-255%5C-255%5C-255%5C-
255%20AND%20name%3A%22pack%2FCOMPANY%20ISO%2FF5%20SSL%20VPN%20endpoint%20veri-
fication%22&range=5184000&decorate=true'
headers = {'Accept': 'application/json'}
apikey = 's3e7xe6lw7cg5tgic9b2e9pq3apeun7wfbzi1jju'
apipw = 'token'
r = requests.get(url, headers=headers, auth=(apikey,apipw))
data = json.loads(r.text)
for x in data['messages']:
        print(x['message']['decorations_hostname'] + " : " + x['mes-
sage']['columns_md5'])
currentDT = datetime.datetime.now()
print ("\nThis file was last updated on " + str(currentDT))
```

Listing 6. Python code

In the listing above, the Python code required to make an API call to Graylog and print the results in a list is shown. The URL and the *apikey* variable are changed for security reasons.

The Python script is ran every 15 minutes by crontab, a job scheduler built into Linux.

```
0,15,30,45 * * * * python /home/topi/graylog.py > /var/www/html/graylog/us-
ers.txt
```

Listing 7. A line in a crontab file

As Listing 7 shows, the Python script is executed on the 0, 15, 30 and 45 minute mark of every hour, every day of the month, every month and every day of the week. The output of the script is piped into a file in the web server directory, overwriting the previously written file. The hashes added to the file would then be read by Sysops and their VPN solution, building a database of authorized Linux VPN users and their machine hashes.

The solution worked perfectly fine, with only a slight delay between enrolling the Linux endpoint to management and getting the required hash for the VPN to work.

4.4.2 Monitoring for security risks

The ISO27001 certification required the Internal IT to monitor the Linux endpoints much more than had been done before. This task actually proved to be the most time-consuming when doing the thesis, as many aspects had to be considered, and the logs provided by Fleet had to be parsed and monitored well.

The monitoring was done completely in Graylog. For it, security dashboards for quick glancing into the status of the fleet could be set up, alongside with more well-refined alarms to immediately inform the Internal IT of more serious security breaches.

The logs were searched and queried by using Elasticsearch. Elasticsearch also allowed the use of regular expression (regex) for the search. Combining them both was extremely powerful in digging through the hundreds of thousands of log entries. For example, the following query in Listing 8 provides an example of using regular expression to search for unauthorized VPN software.

```
source:ip\-255\-255\-255\-255 AND columns_name: /.*[Vv][Pp][Nn].*/ AND NOT
columns_name: /.*[Ff]5.*/ AND NOT columns_name: /.*fsvpn.*/ AND action:
"added"
```

Listing 8. An example of Elasticsearch and regex

The query in the Listing 8 above can be parsed into a few more understandable pieces. The first part of the query is looking for log entries that have a *source* field, which matches the source of the logs sent from Fleet to Graylog. The IP address of the source has been changed for security reasons.

The second part of the query uses regular expression to look into the *columns_name* field of the log entry. For the project multiple scheduled queries were set up in the Fleet to run on multiple different platforms, which reported the installed applications of the endpoint. The queries had the names of the applications in the *columns_name* field. The regex in this example examines the field, checking if it includes the letters V, P and N in succession, either in uppercase or lowercase. This catched all of the applications that had the word VPN in their name. The expression was set up to accept the letters both in uppercase and lowercase since some applications had the word VPN in all caps and

some had it all in lowercase. Regular expression has a flag for case insensitive matching (?i), but I was not able to get it working in Graylog for some reason. This is why it had to be done by using character classes, such as [Vv], accepting either the letter V or v.

The third part of the query was using the *NOT* operator, indicating that if the next part of the query could be matched in the log entry, Graylog should disregard the whole log entry that it was affiliated with. The third part also looks into the *columns_name* field, searching for either the string *f5* or *F5.* This was because the company was using a VPN client provided by a company called F5 Networks, and did not want to see the installations of it in the list of unauthorized VPN clients.

The fourth part is similar to the third, disregarding the installation of *fsvpn*. This was a VPN client provided by our antivirus provider F-Secure.

The fifth part of the query checks the *action* field, indicating that the software that is queried for was added rather than removed. Fleet queries reported both the installed and uninstalled software, of which the latter the Internal IT was not interested in this use case.

After the query had been crafted, a quick value diagram of it could be created and added to a dashboard, as seen on Picture 8.

VPN clients (30d)

Top 50 values		
ExpressVPN.app	54.95%	50
com.expressvpn.ExpressVPN.agent.plist	5.49%	5
openvpn	5.49%	5
network-manager-openvpn-gnome	4.40%	4
network-manager-openvpn	4.40%	4
ExpressVPN	4.40%	4
com.cisco.anyconnect.vpnagentd.plist	3.30%	3
com.purevpn.macapp.HelperTool.plist	3.30%	3
se.ovpn.ovpn-daemon.plist	2.20%	2
ExpressVPN Launcher.app	2.20%	2
com.ovpn.LaunchAtLogin.plist	2.20%	2
ExpressVPN mini.app	2.20%	2
com.fsecure.freedome.vpn.plist	1.10%	1
PureVPN Free VPN Proxy: Unblock with Privacy	1.10%	1
Unlimited Free VPN - Hola	1.10%	1

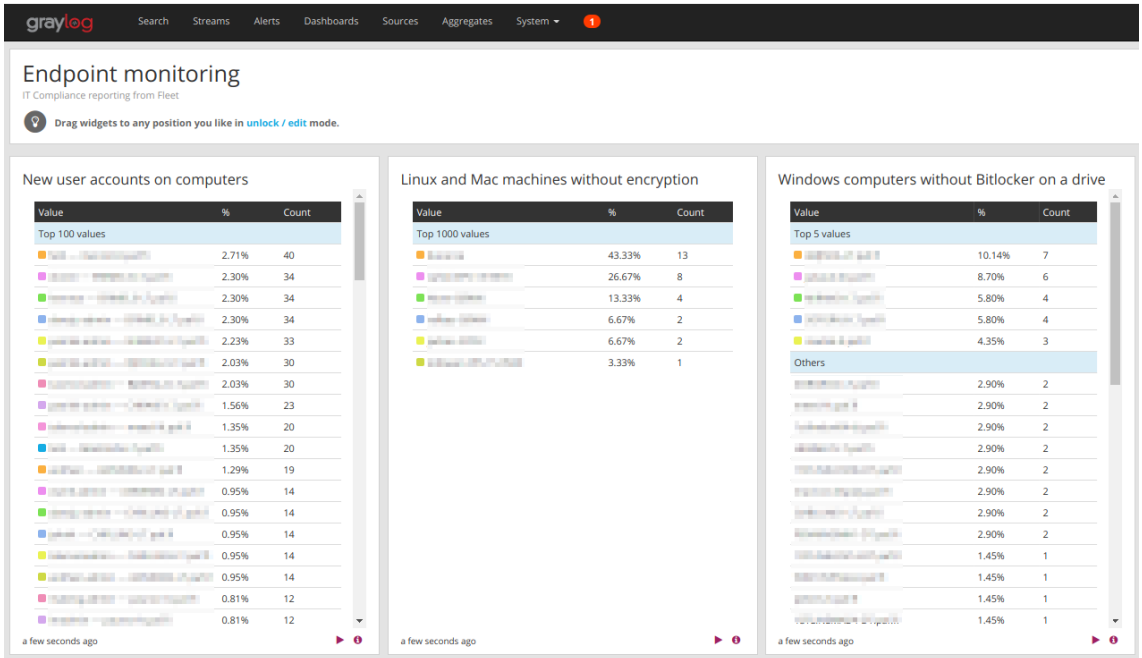
a few seconds ago

Picture 8. A quick value diagram of the query

Graylogs visualization shortcomings could be seen in the picture, as the same applications have been reported multiple times in the chart. This was not a gamebreaking issue for us though, as the idea of the dashboards was to just get a quick glance into the current situation. This could then later be used to investigate the findings more thoroughly.

The dashboards could hold multiple charts, histograms and statistics. The security dashboard that was set up for this project included, for example, a list of new user accounts

on endpoints, endpoints without full disk encryption, Bitlocker status on Windows endpoints, unauthorized VPN clients, Tor Browser installations, Torrent clients, USB writes and more. These are periodically checked by an IT administrator, and actions are taken if they spot anything unusual. The dashboard can be seen in the next picture.

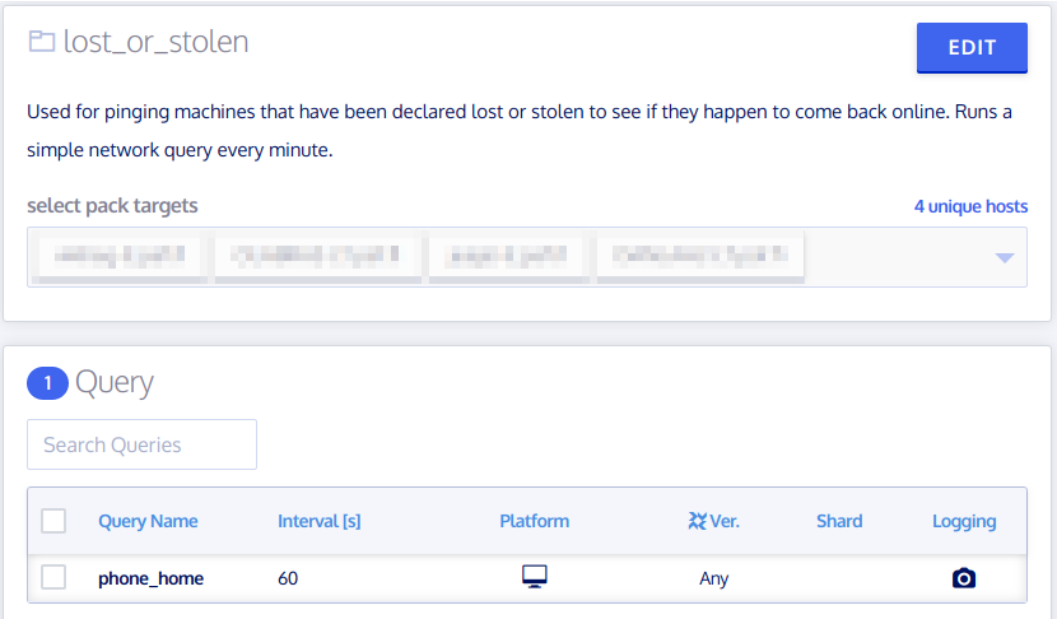


Picture 9. A view of some of the items in the security dashboard

Picture 9 does not include all of the charts set up in the security dashboard, only a couple of examples.

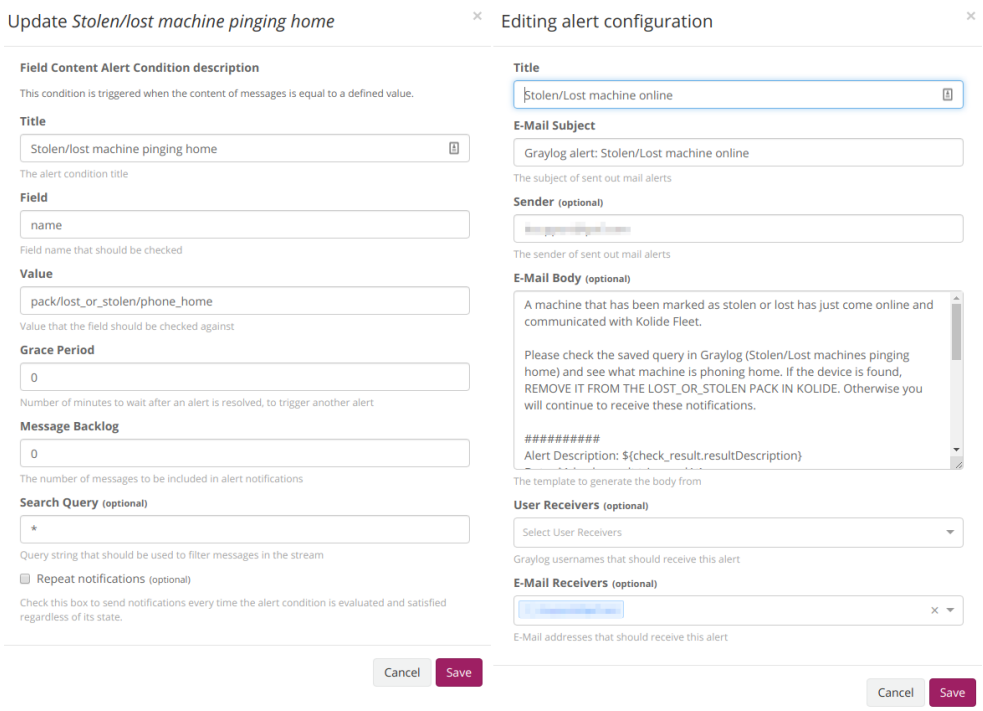
Graylog also supports alerts for more serious and urgent matters, in this case security violations. An example of such setup was an alert for stolen or lost computers coming online.

Fleet allows the user to create query packs for certain hosts only. This feature was used to create a simple query that reports the interface details of an endpoint, since such query could be run on all operating systems without any special modifications. This query was put in a dedicated query pack, with only certain endpoints as targets for it. These endpoints had been reported stolen or missing to the Internal IT, and as such wanted to be monitored for unwarranted use. The query pack can be seen in Picture 10.



Picture 10. A query pack for lost or stolen endpoints

The query in is executed every minute if a lost machine gets connected to the internet. This generates query logs to Graylog, which are then used to trip the alarm. The alert condition definitions and alert notification configurations can be seen in Picture 11.



Picture 11. The alert configuration in Graylog

As seen in Picture 11, the alert gets triggered whenever a log entry is received from the query pack *lost_or_stolen*. This sends out an email alarm to our ticketing system, alerting the ticket duty officer that a computer has possibly been found.

Thus far, the password policies have been strong enough since no misplaced computer has opened a connection back to IT.

5 Summary

The software used in the thesis was fairly unfamiliar to me, and the process of designing and implementing the systems was extremely educational. As the focus of the thesis and the project was in the Linux operating system, I switched my main desktop operating system to Kubuntu some months before the project started, and I am certainly not planning to go back to Windows or Mac in the work environment.

The systems built in this thesis have proven to be highly valuable in the company. The ISO27001 certification dug deep into the information security standards of the company, and Kolide has been used widely to confirm the compliance status of certain aspects, such as disk encryption on Linux systems. Kolide was found to be such a valuable tool for Linux systems that it was quickly taken into use on Mac and Windows systems as well.

Graylog is a great log management platform, and I learned much about it as well. While it did log management very well, visualization is not its strongest suite. We were able to build simple dashboards on top of it, but could have used some advanced utilities as well. Tools like Grafana could have provided those for us, but we did not have enough time to investigate it. Graylog did still provide us with enough functionality to be really useful.

References

1 Stack Overflow. Stack Overflow Developer Survey 2018. [Internet] [cited 2018 November 20]. Available from: https://insights.stackoverflow.com/survey/2018/#technology-developers-primary-operating-systems

2 Juell K. History of Linux | DigitalOcean. [Internet] 2017 October 27 [cited 2018 November 20]. Available from: https://www.digitalocean.com/community/tutorials/brief-history-of-linux

3 GitHub. Ansible. [Internet] [cited 2018 December 7]. Available from: https://github.com/ansible/ansible

4 Kolide. Kolide Fleet. [Internet] [cited 2018 December 7]. Available from: https://kolide.com/fleet

5 GitHub. Osquery. [Internet] [cited 2018 December 7]. Available from: https://github.com/facebook/osquery

6 Graylog. Architecture – Graylog 2.5.0 documentation. [Internet] [cited 2018 December 19]. Available from: http://docs.graylog.org/en/2.5/pages/architecture.html

7 GitHub. fleet/installing_fleet.md at master - kolide/fleet [Internet] [cited 2019 February 14]. Available from: https://github.com/kolide/fleet/blob/master/docs/infrastructure/installing-fleet.md

8 Ansible Documentation. YAML Syntax [Internet] [cited 2019 February 14]. Available from: https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

9 GitHub. fleet/adding-hosts-to-fleet.md at master - kolide/fleet [Internet] [cited 2019 April 8]. Available from: https://github.com/kolide/fleet/blob/master/docs/infrastructure/adding-hosts-to-fleet.md

10 Elastic. Filebeat overview | Filebeat Reference [6.7] [Internet] [cited 2019 April 8]. Available from: https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html

11 IGI Global. What is API [Internet] [cited 2019 April 9]. Available from: https://www.igi-global.com/dictionary/test-template-data-mining-publications/1298