Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Anders Svarverud
Said-Emin Evmurzajev
Sigve Sudland
Sindre Morvik

Cockpit module for administration of Suricata IDS

Bachelor's project in IT-drift og informasjonssikkerhet (BITSEC)
Supervisor: Jia-Chun Lin

May 2021

NTNU
Kunnskap for en bedre verden

Anders Svarverud
Said-Emin Evmurzajev
Sigve Sudland
Sindre Morvik

Cockpit module for administration of Suricata IDS

Bachelor's project in IT-drift og informasjonssikkerhet (BITSEC)
Supervisor: Jia-Chun Lin
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Kunnskap for en bedre verden

Abstract

Section for Digital Security at NTNU has the main responsibility for detection, security analysis and incident response. In an effort to establish a new cyber security center for research and education, this section has entered a collaboration with Uninett AS and UiO CERT. One part of the cyber security center is the ”Sensorplattform” which aims to build the next generation of network based IDS-sensors for the education sector. The main goal of this project is to reduce the tedious administration and operation tasks of Suricata IDS by developing a web-based module based on the administration interface Cockpit. The module enables Suricata administrators to easily start, stop and restart the Suricata service without having to use text-based commands. In addition, our module supports a user-friendly graphical interface for administrating IDS-signatures. Administrators are also able to use our module to easily check the current status of their Suricata services, view the corresponding service logs, and get update on all the alerts generated by Suricata. This thesis begins with background information about Intrusion Detection Systems, Suricata and Cockpit. Further on, we define requirements, explain our development process, and how we designed and implemented the module. Lastly, we discuss the evaluation of the module and conclude with our final thoughts on the project before we discuss what can be improved with future work.

Sammendrag

Seksjon for Digital Sikkerhet ved NTNU har hovedansvaret for deteksjon, sikkerhetsanalyse og hendelsesrespons. I et forsøk på å etablere et nytt datasikkerhetssenter for forskning og utdanning, har seksjonen ingått et samarbeid med Uninett AS og UiO CERT. Èn del av dette datasikkerhetssenteret er «Sensorplattform» som har som mål å bygge den neste generasjon av nettverks baserte IDS-sensorer for utdanningssektoren. Hovedmålet med denne oppgaven er å redusere langtekkelige administrative og operasjonelle oppgaver med Suricata IDS ved å utvikle en web-basert modul basert på administrasjons grensesnittet Cockpit. Modulen tillater Suricata administratorer å enkelt starte, stoppe og restarte Suricata tjenesten uten å måtte bruke tekst-baserte kommandoer. I tillegg til dette vil modulen vår støtte et brukervennlig grafisk grensesnitt for å administrere IDS-signaturer. Administratorer vil også kunne bruke modulen vår til å enkelt se den nåværende status for deres Suricata tjenester, se tilsvarende tjeneste logger, og få oppdatering på alle alarmer som er generert av Suricata. Denne oppgaven begynner med å beskrive bakgrunns informasjon om IDS, Suricata og Cockpit. Videre vil vi definere krav, forklare utviklingsprosessen, og hvordan vi designet og implementerte modulen. Til slutt diskuterer vi evalueringen av modulen og konkluderer med våre tanker rundt prosjektet før vi diskuterer hva som kan forbedres i videre arbeid.

Preface

The Cockpit module for Suricata IDS was developed by four students at Norwegian University of Science and Technology in Gjøvik as a bachelor's thesis during the spring semester in 2021.

We would like to thank Christoffer Vargtass Hallstensen for his help and insight during project meetings.

We would also like to thank Jia-Chun Lin for her excellent guidance and feedback throughout this project.

Contents

Figures

Tables

Code Listings

Acronyms

GPL GNU General Public Licence. 40, 41, 69

GUI Graphical User Interface. 28, 43, 72

IDS Intrusion detection system. 1, 2, 4, 5, 8, 9, 12, 13, 27, 28, 40, 43, 50, 67, 68, 75

MIT Massachusetts Institute of Technology. 40, 41

NTNU Norwegian University of Science and Technology. 2, 24

NTNU SOC Section for digital security at the Norwegian University of Science and Technology. 1, 2, 69

OISF Open Information Security Foundation. 5, 8, 19, 33

UiO CERT University of Oslo Computer Emergency Response Team. 1

Glossary

API Application Programming Interface is a software intermediary that allows two applications to talk to each other. 29, 35, 43, 58, 66

bisect A tool in git that helps find bad commits that introduced a bug in the code. 24

Cockpit Open web-based interface for Linux servers. 1, 3, 8, 10, 27, 28, 29, 31, 33, 35, 37, 40, 41, 43, 58, 66, 75, 76

copyleft The practice of granting the right to freely distribute and modify intellectual property with the requirement that the same rights be preserved in derivative works created from that property. 40

ELK "ELK" is the acronym for three open source projects: Elasticsearch, Logstash and Kibana [1]. 68, 69

GNU/Linux is primarily referred to as the combination of the Linux kernel with the GNU components that form the complete Linux operating system [2]. 1, 8, 10

IDS Sensors Scans the network or host for suspicious and unusual activity. 1

Indicators of compromise Serves as forensics evidence of potential intrusion on a host system or network. 4

Intrusion Detection System A device or software application that monitors a network or systems for malicious activity or policy violations. 1

JavaScript Object Notation JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write, and for machines to parse and generate [3]. 7, 41

Node Package Manager A package manager that manages JavaScript packages. 42

open source Source code that is made freely available for possible modification and redistribution. 1, 5, 8, 24, 40, 42, 43, 69

Suricata Open source intrusion detection system. 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 27, 28, 29, 31, 33, 40, 41, 43, 44, 47, 48, 50, 51, 58, 61, 67, 68, 69, 70, 72, 75, 76

suricata-update A software tool for managing rules, filtering rules and download rules from vendors. 7, 10, 22, 23, 56, 57

symbolic link A term for any file that contains a reference to another file or directory in the form of an absolute or relative path. 41, 42

systemd systemd is a Linux initialization system and service manager that includes features like on-demand starting of daemons, mount and automount point maintenance, snapshot support, and processes tracking using Linux control groups. systemd provides a logging daemon and other tools and utilities to help with common system administration tasks [4]. 2, 27, 29, 35

Ubuntu Open source Linux distribution based on Debian operating system. 2, 10, 41

Uniform Resource Locator A Uniform Resource Locator (URL), colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it [5]. 8

unix socket A UNIX socket, AKA Unix Domain Socket, is an inter-process communication mechanism that allows bidirectional data exchange between processes running on the same machine [6]. 27

Chapter 1

Introduction

This chapter will cover the problem area, project description, target audience, project goal, and the project group.

1.1 Problem Area

With the always present and ever-changing threat landscape in the digital world, there has been an increasing focus on information security in all business sectors. The number of threats an organization has to account for increases yearly, and for larger organizations, it can be increasingly difficult to handle manually. Intrusion Detection System (IDS) is a popular way to automate detection of these threats before they pose a larger problem.

Suricata [7] is an IDS that is popular due to its free and open source nature, multi-threading capabilities allowing for better performance at higher traffic volumes, and support for application layer protocols. Even so, the lack of a graphical interface can make administrating Suricata a daunting task for administrators who are new to the IDS. Cockpit [8] is a web-based interface for GNU/Linux, and it is designed to help administrate a server with a web browser. Cockpit facilitates the creation of custom modules which can be added to its interface to provide a better user experience. However, to our best knowledge, there is no such module for Suricata at the time of conducting this bachelor project.

1.2 Project Description and Goal

This project was given by the Section for digital security at the Norwegian University of Science and Technology in Gjøvik as part of a collaboration with Uninett AS and University of Oslo Computer Emergency Response Team. The collaboration aims to establish a national cyber security center for the education sector named ”Cybersikkerhetssenteret for Forskning og Utdanning”.

The objective of this project is to develop a module for Cockpit to further simplify the administration of Suricata IDS. The module is requested for the ”Sensorplattform” which aims to be the next generation of network based IDS Sensors for the education sector, and is a part of the larger collaboration.

First of all, the module should allow administrators to start, stop and restart the Suricata IDS service via the web interface, and the interface should display the current status of the IDS service. Another feature of the module is the ability to administrate IDS-signatures. This means that administrators can add existing or custom signatures to their own Suricata and manage these signatures (such as editing, deleting, enabling and disabling) from the web interface.

Secondly, the module should offer functionality for adding and removing signature vendors. Note that a signature vendor is a third party providing different rule sets for Suricata IDS. A rule set is a list of signatures used to match against when searching the network traffic for suspicious activity.

Lastly, the module should also be able to display relevant service logs to make troubleshooting easier. All alerts issued by Suricata should be displayed properly, and can be sorted by different criteria e.g., severity, source IP, destination IP, etc. All in all the module aims to be a "one stop dashboard" for administrating Suricata, and lower the entry barriers for administrators to easily perform administrative tasks on Suricata IDS sensors in their own organizations.

1.3 Target Audience

The project aims to lower the entry barriers for end users by offering a graphical interface that can be accessed in a browser for Suricata IDS. Here the end users will typically be network and system administrators who are relatively familiar with the administration of Suricata. The project could also be of use to organizations who are in need of a more intuitive way to administrate Suricata.

1.4 Limitations

To streamline the development of the module, we have decided to use Ubuntu as our development platform since it is one of the most commonly used Linux distributions [9]. We used the system and service manager systemd for information regarding the status of services. This might cause some troubles on more obscure Linux distributions that do not support systemd, but it should not be a problem on the most popular distributions.

Furthermore the module will be more focused on the administrative parts of Suricata, and less on the actual monitoring of alerts. As such, alerts will be displayed on the web interface in a simple list.

1.5 Project Group

The project was requested by Christoffer Vargtass Hallstensen, group leader at NTNU SOC, and Arne Øslebø, Senior Technical Architect working for Uninett AS. Jia-Chun Lin, assistant professor at NTNU, is our supervisor for the project.

The group consists of four bachelor students from IT-Operations and Information Security (BITSEC). From the studies the group have gained knowledge in programming, networks, security, risk management and IT-operations. The bachelor project required us to learn more about web-development with focus on HTML, CSS, JavaScript, React and UX/UI-design. Additionally the group had to learn about Cockpit and Suricata.

1.6 Thesis Structure

- **Chapter 1 - Introduction:** Description of project, problem area, target audience, project goal and project group.
- **Chapter 2 - Background:** This chapter will briefly explain all technologies and terminologies used in the thesis.
- **Chapter 3 - Requirements:** In this chapter, we present the functional and non-functional requirements as well as all use cases for the module.
- **Chapter 4 - Development Process:** Discussion around development model and explanation of the workflow.
- **Chapter 5 - Implementation:** Explaining the development environment used in the project.
- **Chapter 6 - Technical Design:** This chapter covers the design of the graphical user interface and the functions of the module.
- **Chapter 7 - Evaluation:** Evaluation of the module against the functional requirements and user feedback.
- **Chapter 8 - Closing Remarks:** Conclusion, discussion and learning outcome for the project, and suggestions for future improvements.

Chapter 2

Background

This chapter aims to give some background knowledge regarding all technologies and terminology used in this thesis.

2.1 Intrusion Detection System

An Intrusion detection system (IDS) is a device or software that observes a network for unwanted traffic and notifies administrators of the network activity by issuing an alert [10]. An IDS is usually either network-based or host-based and use either a signature-based or an anomaly-based detection method. Each of these types of IDSs are briefly introduced below.

Network-based IDS
With a network-based IDS, sensors are usually placed on strategic locations in the network to monitor traffic to and from devices on the network. If traffic on the network matches to a library of known attacks, or the IDS identifies unusual behavior or network traffic, an alert can be sent to the administrator. One use case could be to place the IDS on the same subnet as the firewalls to monitor if someone attempts to attack them [10].

Host-based IDS
A host-based IDS is placed on an individual host or device and will match against a snapshot of important system files. If for example a word-processor starts exhibiting strange behaviours like modifying the systems password database, a host-based IDS might pick up on it, and issue an alert so that a system administrator is made aware [11].

Signature-based IDS
A signature-based IDS usually monitors inbound network traffic for attacks by comparing the traffic with a list of known Indicators of compromise. The list might consist of specific network attack behaviours, known byte sequences, malicious domains, email subject lines and file hashes. Specific sequences and patterns matching an attack signature may be found within network packet headers, in sequences of data matching known malware or malicious patterns, in destination or source network addresses, or in specific sequences of data or packets [12]. A significant drawback of a signature-based IDS is that it does not detect unknown attacks e.g., zero-day attacks.

Anomaly-based IDS
Anomaly-based IDS is used to detect attacks by comparing incoming traffic to a model of trusted activity and anything not found in the model is declared suspicious. Machine learning is often used to create such models, and these models can be trained according to applications and hardware configurations [10].

2.2 Suricata

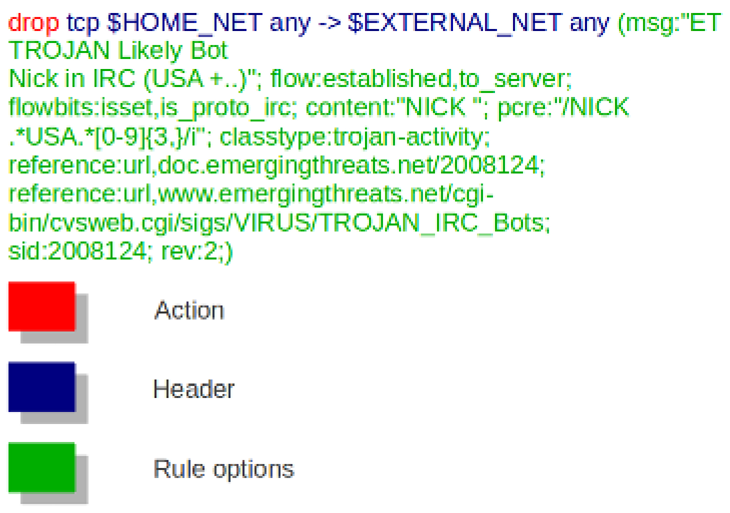
Suricata [7] is a free and open source network threat detection engine that provides intrusion detection, intrusion prevention and network security monitoring. The Suricata project is community driven and focuses on security, usability and efficiency, and is owned and supported by the Open Information Security Foundation (OISF). Suricata differentiates itself from other similar network threat detection engines like Snort [1] by providing multi-threading which allows for better performances [13]. Additionally the rule language of Suricata makes it easier to match conditions in the application layer protocol without a deeper understanding of packet and protocol structure.

Figure 2.1: The Suricata logo

2.2.1 Signatures

Signatures or rules are an important aspect of Suricata as this is what the IDS uses to detect suspicious network activities. An administrator can use existing rule sets such as the one provided by OISF [2], but it is also possible to make custom signatures or modify existing ones [14]. Figure 2.2 shows an example of a signature. The first part in red signifies an action to be taken if a network packet matches the header represented in blue. The rule options in green are parameters that can be added to the signature for deeper and more customized detection. More details about action, header and rule options are described below.

[1]https://www.snort.org/
[2]https://openinfosecfoundation.org/rules/trafficid/trafficid.rules



```
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET
TROJAN Likely Bot
Nick in IRC (USA +..)"; flow:established,to_server;
flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK
.*USA.*[0-9]{3,}/i"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

🟥 Action

🟦 Header

🟩 Rule options

Figure 2.2: An example of a Suricata signature [15]

Action
Action is a property of the signature which decides what will happen when a network packet matches a signature. An action can be one of the following four [16]:

- **Pass:** Suricata breaks off the scanning of the packet, and skips to the end of all the rules for the current packet.
- **Drop:** Immediately stops the packet from being transmitted further. A corresponding alert will be generated.
- **Reject:** The packet is rejected. The receiver and sender will receive a reject packet response. An alert will be generated.
- **Alert:** The packet is treated like any other non-threatening packet, with the exception that an alert will be generated by Suricata for the system administrators to examine.

Header
A header is composed by the protocol to match against, the source IP address and port number, the destination IP address and port number, and a direction that says something about which way the signature has to match. In Figure 2.2 we match against packets with the protocol *tcp*, source IP address of *$HOME_NET* and source port of *any*. The direction is a right arrow, this means only packets going from the source to destination will be matched. To match it both ways we would need to replace "->" with "<->".

The destination to match against has an IP address *$EXTERNAL_NET* and a port of *any*. *$HOME_NET* and *$EXTERNAL_NET* are variables defined in "suricata.yaml" which contain IP addresses. If we want to match against other packets with different properties, we need to change the values in the different fields of the header so they correlate [14].

Rule Options
Rule options defines the specifics of a rule. The rule options are closed within a parenthesis, and

each option is separated by a semicolon. Options usually contains a keyword separated by settings, however some options only need the keyword [17]. In Figure 2.2 the green text represents the rule options, and the first option uses the *msg* keyword with the text as shown below.

```
msg:"ET TROJAN Likely Bot Nick in IRC(USA +..)";
```

This specific option will give information about the signature and the possible alert [18].

2.2.2 Suricata files

This section will give a brief overview of the important Suricata files. Some of these files are used for configuration, while others contains logs and alerts.

- **suricata.yaml** Suricata uses a file named ”suricata.yaml” for its configuration [19]. In this file one can for example configure network settings and the type of output to generate.

- **eve.json** Suricata provides an output facility called EVE that outputs all alerts, anomalies, metadata, file info and protocol specific records through JavaScript Object Notation. This output can be accessed through the ”eve.json” file. For example if you want information about an alert that happened in a specific time frame you can find it in ”eve.json” [20].

- **stats.log** Interesting statistics regarding Suricata such as memory usage and packet loss can be found in the ”stats.log” file. By default the statistics will be produced every 8 seconds and appended to ”stats.log” [21]. However it is possible to change the interval at which statistics are produced, and to overwrite ”stats.log” instead of appending new data to it. These changes have to be made in the ”suricata.yaml” file [22].

- **suricata.rules** All rules are by default merged into a single file called ”suricata.rules” [23].

- **update.yaml** Suricata has a rule download and management tool called suricata-update. When this command is invoked it reads configuration from a file which by default is called ”update.yaml”. This file specifies the path to the different rule filter configuration files as well as local and remote sources to be used [24].

- **suricata-update configuration files** There are four suricata-update config files which by default are called ”disable.conf”, ”enable.conf”, ”drop.conf” and ”modify.conf” [25]. In these files we specify the rules to apply the filters to by using rule matching specifiers. Rules can be matched using the signature ID, filename, rule group or regular expressions [26]. The functions of the different rule filter configuration files are [27]:
    - **disable.conf:** Rules matching the specifiers in this file will be disabled
    - **enable.conf:** Rules matching the specifiers in this file will be enabled
    - **drop.conf:** Rules matching the specifiers in this file will be converted to drop rules. Packets that match the drop rules will be dropped, and an alert will be generated.
    - **modify.conf:** Rules matching the specifiers in this file will be modified. A modification can for example change rule action from drop to alert.

2.2.3 Vendors

Suricata IDS provides the functionality to add and use rule sets from different vendors. In this thesis, we categorize vendors into *custom vendors* and *default vendors*, and we define them as follows:

- Custom vendors are manually added by providing Uniform Resource Locators to the custom vendors' source files.
- Default vendors are the vendors defined in the "index.yaml" source [3] file provided by OISF and comes with Suricata by default.

Some vendors are not publicly available and require a *secret code*. Without the secret code the vendor source files will not be usable by Suricata.

2.3 Cockpit

Cockpit [8] is an open source web-based interface for managing and monitoring GNU/Linux servers. It is included in most of the major distributions and can be installed from the distributions' package manager. Cockpit's graphical interface makes it a good entry point for new Linux users while remaining useful for more experienced users. With Cockpit, an administrator can accomplish a large range of tasks such as starting containers, administer storage, configure networks, inspecting logs and more [8].

Additionally the Cockpit team supports the creation of custom modules using Cockpit's own interface. To help kick-start the process and encourage the building of new modules they have created the Cockpit Starter Kit [4]. There is already a long list of optional and third-party applications [5] that can easily be added to the Cockpit interface.

COCKPIT

Figure 2.3: The Cockpit logo

[3]https://www.openinfosecfoundation.org/rules/index.yaml
[4]https://cockpit-project.org/blog/cockpit-starter-kit.html
[5]https://cockpit-project.org/applications.html

Chapter 3

Requirements

This chapter discusses the functional requirements, additional features that we decided to implement, non-functional requirements, and use cases of the module.

3.1 Functional Requirements

In this section we introduce all the functional requirements requested by the client. The module must satisfy the following requirements:

1. The module must allow the administrators to start, stop and restart the Suricata service via the click of the respective buttons on a web interface.
2. The module should allow the administrators to view IDS signatures.
3. The module must enable administration of IDS signatures. The administrators should be able to add signatures by uploading local files or creating new files. The administrators should be able to edit or delete signatures.
4. The module must allow the administrators to download IDS signatures from a vendor and use them.
5. The module must display logs related to the Suricata service.
6. The current status of Suricata service should be displayed.
7. The module must display the alerts generated by the Suricata IDS. The alerts can be sorted by date and time, priority, protocol, category and id.
8. The module must be licensed as open source.

3.2 Additional Features

In this section we introduce all the additional features that were added during development of the module. Some of the additional features were suggested by the client while others came naturally during development.

1. The module allows the administrators to search and sort log entries using the fields of a log entry (e.g., date and time).

2. The module allows the administrators to manage vendors that requires secret code.
3. The module allows the administrators to configure Suricata and suricata-update.
4. Changes made to the ”suricata.yaml” and ”update.yaml” files are checked for valid YAML syntax.
5. Edit the suricata-update configuration files (described in 2.2.2) which are used for modifying, dropping, disabling and enabling rules in rule sets obtained from vendors.

3.3 Non-functional Requirements

This section defines a set of non-functional requirements that are used to specify the quality attributes of the module.

3.3.1 Compatibility

The module will be developed and tested with the Ubuntu operating system, but it is expected to work with many other distributions. This module is developed for Suricata v6.0.1, but might work with older and newer versions of Suricata as well. Cockpit is developed for and routinely tested with Mozilla Firefox, Google Chrome and Microsoft Edge. According to the documentation [28], Cockpit *might* also work with other browsers. With this in mind we take the same approach and focus our development for those three browsers.

3.3.2 Reliability and maintainability

Taking a modular approach and following coding standards ensures maintainability of the module. The module is separated into smaller pieces in order to make the job of finding and fixing bugs in the code easier. This will also help increase the readability of the code. The airbnb [1] style guide is followed and together with ESLint this will reduce the chance of making mistakes and introducing bugs in the code.

3.3.3 Usability

The goal of the module is to make it easy to use. In other words, end users should be able to administrate Suricata with minimal knowledge of the Suricata service. The interface of Cockpit is a good starting point and we attempt to design our module in a way that blends well with that. An intuitive graphical interface will go a long way to ease up the demanding technological knowledge otherwise needed to administrate Suricata through a GNU/Linux terminal. We will implement input validation to ensure that valid values are entered and provide suitable error messages. Some detrimental changes, e.g., deleting a file through the module, will prompt the administrator with a confirmation dialogue. To further increase the usability, we will add tool tips to explain important elements.

[1]https://github.com/airbnb/javascript

3.4 Use Cases

Figure 3.1 represents the system administrator's interaction with the module in form of use cases. For example from the figure we can interpret that the system administrator can start the Suricata service from the web interface.

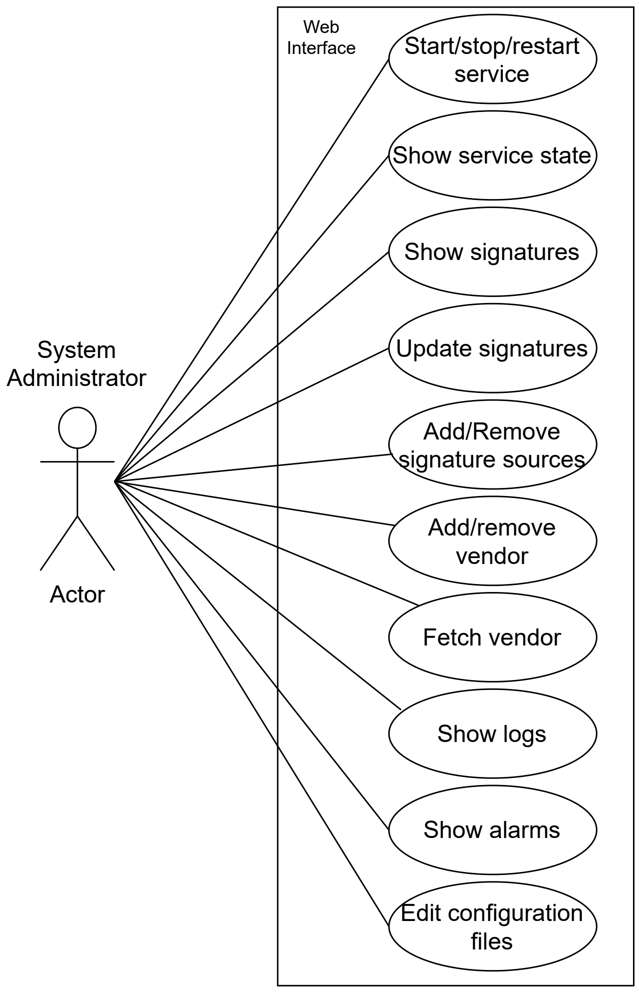


Figure 3.1: Use Case Diagram of our module

In Figure 3.1 to 3.16 a more detailed description of each use case is presented. The detailed Use Cases explain the steps an administrator has to take to achieve the desired goal.

Table 3.1: Start service

Start service	
Actor:	System Administrator
Goal:	Start the Suricata service
Description:	The system administrator presses the ”Start” button located under the ”Service” tab. The Suricata IDS service will be started.
Pre-condition:	Service tab is active
Sequence of Events:	Press ”Start” button

Table 3.2: Stop service

Stop service	
Actor:	System Administrator
Goal:	Stop the Suricata service
Description:	The system administrator presses the ”Stop” button located under the ”Service” tab. The Suricata IDS service will be stopped.
Pre-condition:	Service tab is active
Sequence of Events:	Press ”Stop” button

Table 3.3: Restart service

Restart service	
Actor:	System Administrator
Goal:	Restart the Suricata service
Description:	The system administrator presses the ”Restart” button located under the ”Service” tab. The Suricata IDS service will be restarted.
Pre-condition:	Service tab is active
Sequence of Events:	Press "Restart" button

Table 3.4: Add signature file

Add signature file	
Actor:	System Administrator
Goal:	Add signatures to Suricata
Description:	The system administrator adds a rule set to the signature list
Pre-condition:	Signatures tab is active
Sequence of Events:	1. Press ”Local rules” subtab 2. Press ”File” dropdown menu 3. Select ”Upload file” 4. Drag and drop a file into the popup or browse the file system. 5. Press ”Upload” 6. Press ”Apply changes” to update and reload Suricata.
Alternative Events:	3. Select ”Create file” 4. Enter a name for the new file 5. Press ”Create” 6. To add content to the file, the file must be edited, see use case 3.6.

Table 3.5: Delete signature file

Delete signature file	
Actor:	System Administrator
Goal:	Delete a signature file
Description:	The system administrator deletes a rule set from the signature list
Pre-condition:	Signatures tab is active
Sequence of Events:	1. Press ”Local rules” subtab 2. Press the ”Action” dropdown menu besides the file to be deleted. 3. Select ”Delete file” 4. Press ”Yes” to confirm the action 5. Press ”Apply changes” to update and reload Suricata.
Alternative Events:	4. Press ”No” to deny the action

Table 3.6: Edit signature file

Edit signature file	
Actor:	System Administrator
Goal:	Edit signature file
Description:	The system administrator edits a rule set from the signature list
Pre-condition:	Signatures tab is active
Sequence of Events:	1. Press the ”Local rules” subtab 2. Press the ”Action” dropdown menu besides the file to be deleted. 3. Select ”Edit file” 4. Make changes to the file 5. Press ”Save changes” 6. Press ”Apply changes” to update and reload Suricata.

Table 3.7: Add custom vendor

Add custom vendor	
Actor:	System Administrator
Goal:	Add signature source
Description:	Add signature sources from vendors
Pre-condition:	Signatures tab is active
Sequence of Events:	1. Press the ”Add custom vendor” button 2. Fill in the name of the vendor 3. Fill in the source URL for the vendor 4. Press ”Submit” 5. Press ”Apply changes” to update and reload Suricata.

Table 3.8: Remove custom vendor

Remove custom vendor	
Actor:	System Administrator
Goal:	Remove custom vendors
Description:	The system administrator can remove custom vendors. Note that default vendors can not be removed.
Pre-condition:	The ”Signatures” tab is active
Sequence of Events:	Press ”Remove” on the vendor

Table 3.9: Delete secret code

Delete secret code	
Actor:	System Administrator
Goal:	Delete secret code
Description:	The system administrator can delete secret code that are required for some vendors
Pre-condition:	1. Signatures tab is active 2. A secret code has been stored
Sequence of Events:	1. Find the specific vendor on the list 2. Press ”delete secret code”

Table 3.10: Fetch vendors

	Fetch vendors
Actor:	System Administrator
Goal:	Fetch the default vendors
Description:	OISF provides an index [29] of default vendor sources. If the index is updated the administrator can fetch the default vendors
Pre-condition:	Signatures tab is active
Sequence of Events:	Press the ”Fetch vendors” button

Table 3.11: Apply changes

	Apply changes
Actor:	System Administrator
Goal:	Apply changes made to the signatures
Description:	The system administrator presses the *Apply changes* button on the *Signatures* tab after making changes to the signatures. The signatures will be updated and reloaded.
Pre-condition:	Signatures tab is active
Sequence of Events:	Press ”Apply changes”

Table 3.12: Show logs

Show logs	
Actor:	System Administrator
Goal:	Monitor logs relevant to the Suricata service.
Description:	The system administrator presses the ”Logs” tab to display all relevant logs in real-time or from a specific time period.
Pre-condition:	Logs tab is active
Sequence of Events:	Press ”suricata.service” subtab
Alternative Events:	Press ”stats.log” subtab

Table 3.13: Show alerts

Show alerts	
Actor:	System Administrator
Goal:	Monitor alerts
Description:	The system administrator monitors the alerts generated by Suricata. The total amount of alerts triggered is also displayed, as well as the amount of times each individual alert has been triggered.
Pre-condition:	Alerts tab is active

Table 3.14: Edit suricata.yaml

Edit suricata.yaml	
Actor:	System Administrator
Goal:	Configure Suricata
Description:	The system administrator edits the ”suricata.yaml” configuration file
Pre-condition:	Config tab is active
Sequence of Events:	1. Press ”suricata.yaml” subtab 2. Find property to edit 3. Edit the selected properties of suricata.yaml 4. Save changes
Alternative Events:	2. Press ”edit directly” button to edit suricata.yaml directly

Table 3.15: Edit update.yaml

Edit update.yaml	
Actor:	System Administrator
Goal:	Configure suricata-update
Description:	The system administrator edits update.yaml configuration file
Pre-condition:	Config tab is active
Sequence of Events:	1. Press ”update.yaml” subtab 2. Find property to edit 3. Edit the selected properties of update.yaml 4. Save changes
Alternative Events:	2. Press ”edit directly” button to edit update.yaml directly

Table 3.16: Edit suricata-update config files

Edit suricata-update config files	
Actor:	System Administrator
Goal:	Configure suricata-update files
Description:	The system administrator edits suricata-update configuration files
Pre-condition:	Config tab is active
Sequence of Events:	1. Press ”suricata-update config files” subtab 2. Open the dropdown menu 3. Select a configuration file to edit 4. Edit file 5. Press ”Save changes”

Chapter 4

Development Process

This chapter describes the development model used for developing the module, and the tools and processes used for managing our workflow.

4.1 Development Model

The group decided on using Kanban [30]. Kanban's agile nature allows us to operate with a continuous workflow and optimize the work being done by capitalizing on each group member's strengths. Kanban gives us a better overview of our progress and makes for easier task delegation, as each member mainly choose tasks themselves.

4.2 Documentation

This section discusses the tools and processes used to document our work.

Work Management
Kanban was used as a workflow management method to structure and assign tasks in the group. This approach helps us organize and manage tasks more efficiently, and will help us recognize bottlenecks during development.

Thesis writing
The thesis will be written using the software system LaTeX commonly used to write scientific documents in academia. We use the template that [1] NTNU provides for bachelor, masters and PhD theses. To create the diagrams in the thesis we will be using the free online diagram software ”diagrams.net” (formerly draw.io) [2].

Source code
The source code of the project will be managed and tracked with Git [31] which is a free open source version control system for projects of all sizes. Git helps us easily bisect bad code that has

[1]https://github.com/COPCSE-NTNU/thesis-NTNU
[2]https://app.diagrams.net/

been pushed.

Meetings
Each Wednesday we had status meetings with our supervisor where we presented our weekly progress and our plans for the next week. In the meetings we were given feedback on the progress and tips for how to proceed. Additionally we had meetings with our client to make sure we were on the right track with what they had envisioned. The group met three times per week over Discord to work collectively on the project.

Meeting minutes
Minutes from each meeting were kept to make sure no information was lost during the project. Everyone was responsible for writing notes about what they had done during the week. For the weekly status meetings with our supervisor, we prepared slides detailing what had been done, what we wanted help with, and any challenges we faced.

Time management
The time management software Clockify [3] was used to keep track of the time each member spent on the project. The time tracking was further split into different tasks to see where the time was spent.

4.3 Workflow

There were five roles used within the group:

Group Leader: Sindre Morvik (vara: Sigve Sudland)
Responsible for structuring the workload and making sure everyone knew what tasks needed to be done.
Secretary: Anders Svarverud, Said-Emin Evmurzajev
Responsible for taking notes during meetings with our supervisor and client as well as writing a short summary of the work done each week.
Overleaf technician: Sigve Sudland
Responsible for the LaTeX document.
Thesis leader: Sindre Morvik
Responsible for overseeing the writing of the thesis.
Development leader: Sigve Sudland
Responsible for overseeing the work on the module.

The group decided on a few rules and routines to follow in order to have some transparency about what is expected of each member. We kept track of the time used by each member by using the time tracking application Clockify. This was done in part to include a time sheet in the thesis and to make sure that each member kept within the expected weekly hours. This allowed us to make sure that our delegation of tasks was fair and that overtime was not a regular occurrence.

[3]https://clockify.me

To manage the tasks during the project period, we used the online tool Trello [4] to make a Kanban board. All the tasks related to the project were added to the board, and members could pick and choose whichever task they felt most comfortable with. This could lead to some tasks not being picked because no member felt comfortable doing them. To make sure no tasks were left in limbo too long we had regular meetings to discuss the workload with an emphasis on making sure that members could work with what they wanted, as long as that was possible.

We used the VoIP-program Discord for our main communication platform. Most meetings the group had were organized and carried out here. Members were expected to keep a close eye and be available on Discord during regular working hours. For communication and meetings with our supervisor and client we used Microsoft Teams.

Git, a distributed version control system, was used to keep track of the development of the module and to allow us to work on different tasks from the same code base.

[4]https://trello.com

Chapter 5

Technical Design

This chapter aims to explain the design and functionality of our Cockpit module for Suricata IDS. The chapter will start by describing the overall system architecture before going into more detail about the module architecture.

5.1 System Architecture

Figure 5.1 shows the overall system architecture of the Cockpit module for Suricata. Cockpit and Suricata are both installed on the same Linux machine and uses systemd services to request the status of the Suricata service, and to start, stop or restart Suricata. Cockpit will use the unix socket to reload the signatures when they have been updated.

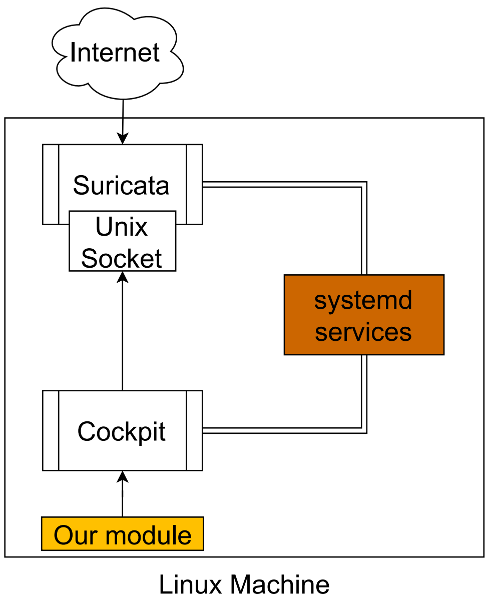


Figure 5.1: Overall system architecture diagram of the module

5.2 Module Architecture

The module is implemented as a web user interface with five tabs namely *Service*, *Signatures*, *Logs*, *Alerts*, and *Config*. Figure 5.2 illustrates the architecture of the module. The different tabs of the module are colored grey while information displayed on the tabs are represented in purple. All buttons of the module are displayed with a blue color in the figure, and Linux commands used for some of the functionality are represented as brown. The cases where the module accesses files is colored green.

The *Service* tab displays the current state of the Suricata IDS service. In addition, this tab provides three buttons to start, stop and restart Suricata. The *Signatures* tab allows system administrators to manage signature sources and update rule sets. The *Logs* tab shows relevant service logs from the Suricata IDS service and the *Alerts* tab displays alerts issued by Suricata. The *Config* tab allows the system administrator to configure Suricata through Cockpit's GUI.

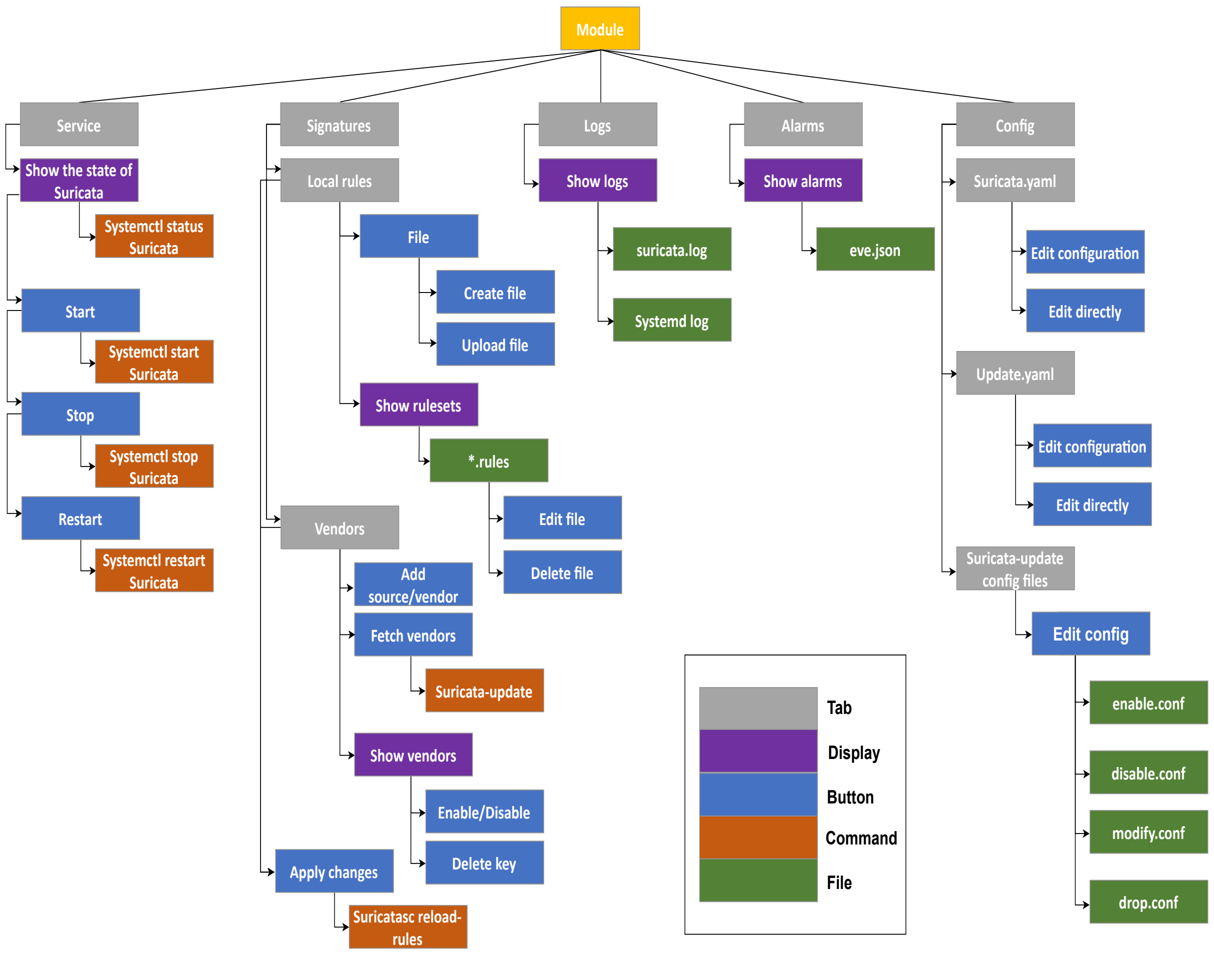


Figure 5.2: The architecture diagram of our module

5.3 Sequence Diagram

This section will depict the involved objects and the interactions between them to carry out the functionality for each tab of the module.

5.3.1 The Service tab

Figure 5.3 shows the sequence diagram for the service tab of the module. The diagram accounts for all functionality available on the tab. Once the service tab is selected, the Cockpit API [32] sends a DBus message to systemd in order to start monitoring the *suricata.service* unit on the server in real-time. When a property has been updated on the service, systemd returns the selected properties to Cockpit before an update function displays the new data on the service tab. After this the service calls the Cockpit API spawn function to execute the ”suricata --build” command in order to obtain build info from Suricata. The Suricata version is then displayed on the service tab while the rest of the result from the ”suricata --build” command is filtered out.

In the next steps the Cockpit API for file access is used to read and return the contents of *suricata.rules* file. The service tab then displays all active signatures by counting the lines in *suricata.rules*. In the last steps the user can choose to start, stop or restart the Suricata process. The sequence being run is mostly identical no matter what choice the user makes except for one string in the call argument. The service uses Cockpit API to send a "DBus" message to systemd describing how to handle the service. In the next run-through of the sequences Cockpit will be notified of the status of this operation thanks to the monitoring loop in step three and update the data that way.

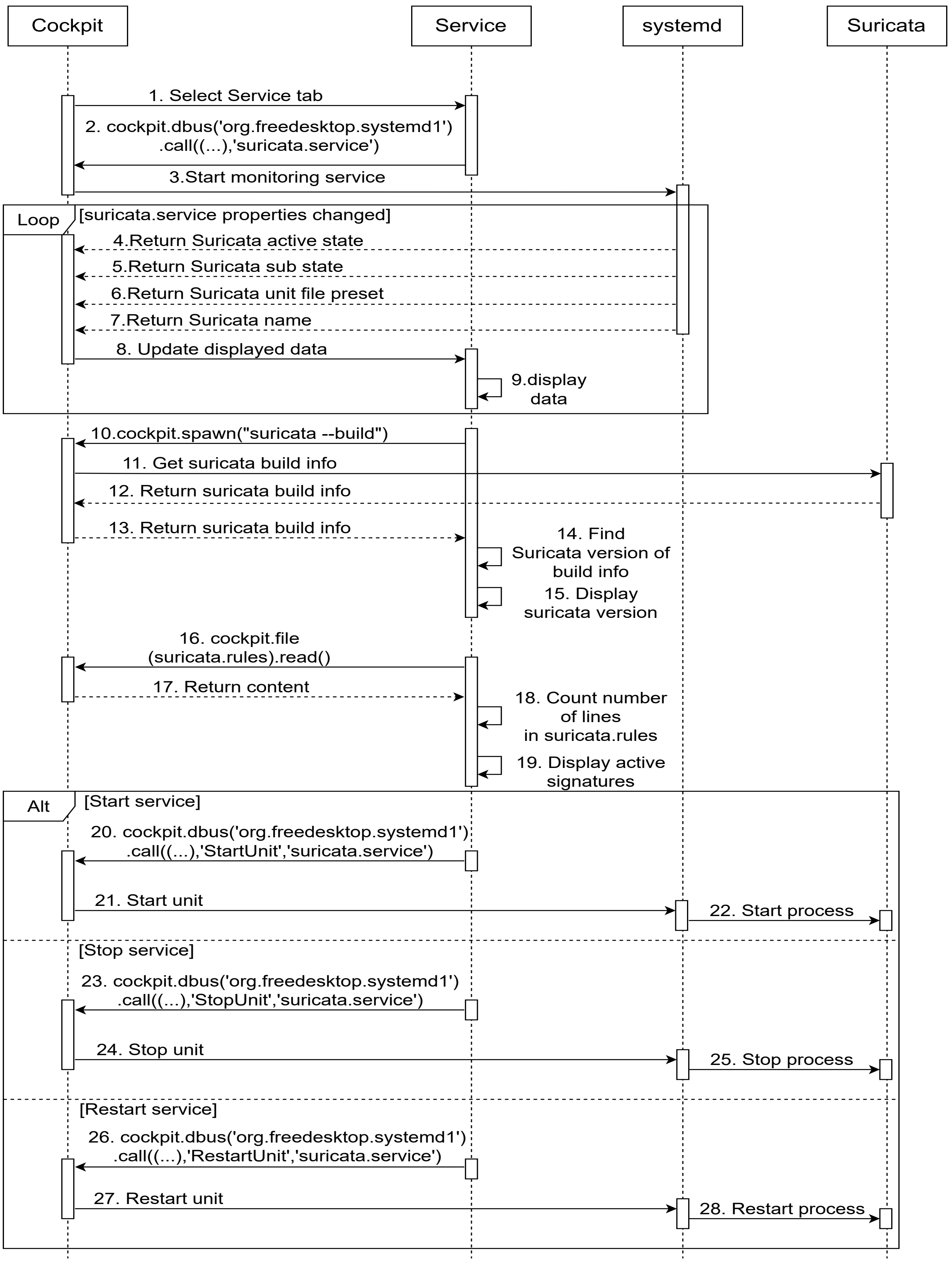

Cockpit
Service
systemd
Suricata
1. Select Service tab
2. cockpit.dbus('org.freedesktop.systemd1') .call((...),'suricata.service')
3.Start monitoring service
Loop
[suricata.service properties changed]
4.Return Suricata active state
5.Return Suricata sub state
6.Return Suricata unit file preset
7.Return Suricata name
8. Update displayed data
9.display data
10.cockpit.spawn("suricata --build")
11. Get suricata build info
12. Return suricata build info
13. Return suricata build info
14. Find Suricata version of build info
15. Display suricata version
16. cockpit.file (suricata.rules).read()
17. Return content
18. Count number of lines in suricata.rules
19. Display active signatures
Alt
[Start service]
20. cockpit.dbus('org.freedesktop.systemd1') .call((...),'StartUnit','suricata.service')
21. Start unit
22. Start process
[Stop service]
23. cockpit.dbus('org.freedesktop.systemd1') .call((...),'StopUnit','suricata.service')
24. Stop unit
25. Stop process
[Restart service]
26. cockpit.dbus('org.freedesktop.systemd1') .call((...),'RestartUnit','suricata.service')
27. Restart unit
28. Restart process

Figure 5.3: Sequence Diagram for the Service tab

5.3.2 The Signatures tab

Figure 5.4 shows the sequence diagram for "Local rules" subtab, which is the default selected subtab on the signatures tab. Cockpit's spawn function is called with ”ls /etc/suricata/rules” which returns and displays the local signatures in a sortable list on the subtab. The next step depends on whether the administrator chooses to ”create file”, ”upload file”, ”edit file”, ”delete file” or ”apply changes”.

To create a file the *cockpit.spawn(”touch ${fileName}”)* is called where *fileName* is input from the administrator. To upload, edit or delete a file, the *cockpit.file().modify()* function is called with the *fileName* variable as argument. ”Apply changes” will spawn a process with the command *suricata-update* that checks if any changes have been made. Once this is done the Suricata service is reloaded in order to make use of the new changes. The output of the process will then be returned to the subtab and shown to the administrator. Finally, the *cockpit.spawn()* function is called one last time to display all the files in the */etc/suricata/rules* directory.

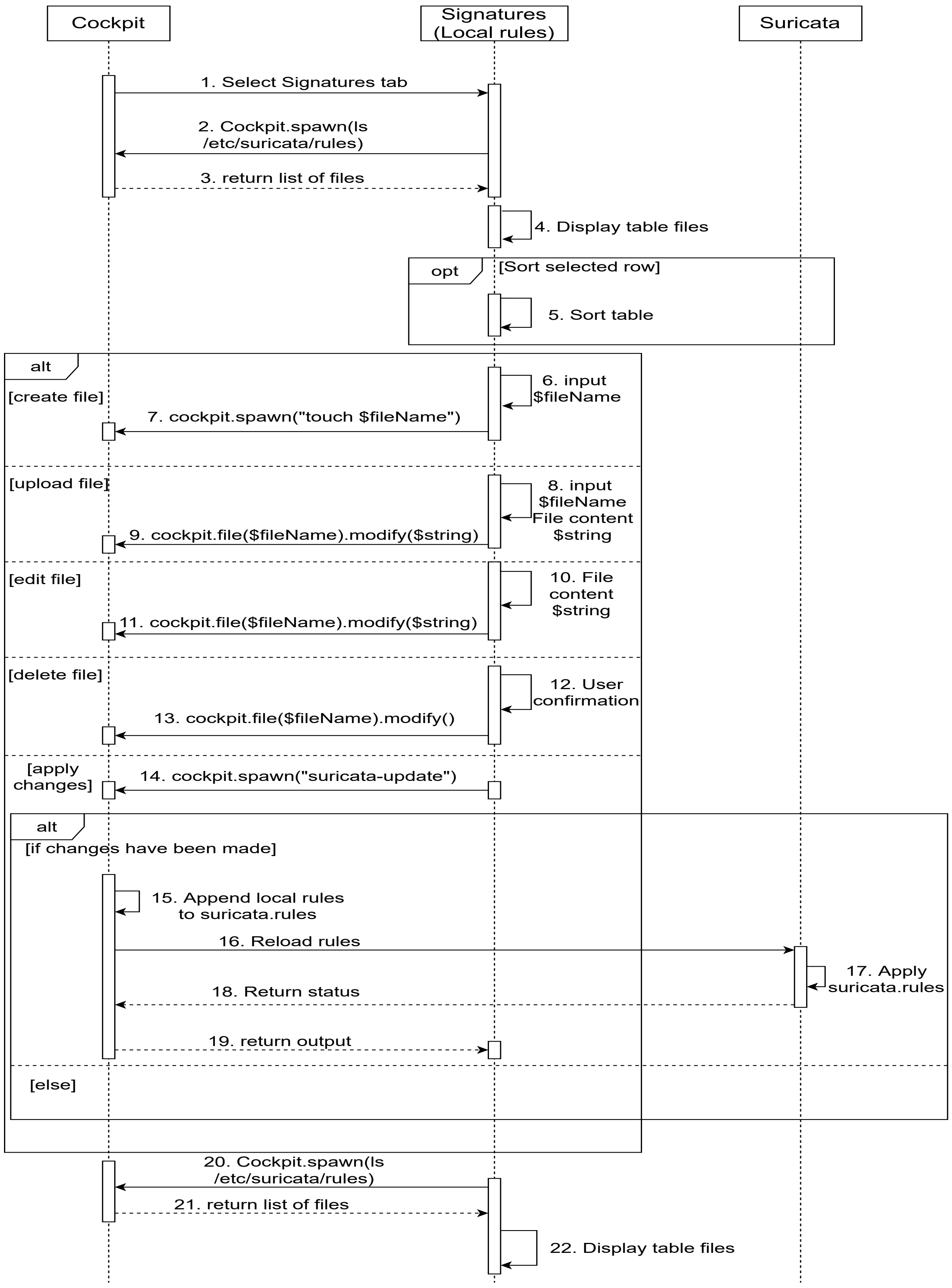


Figure 5.4: Sequence Diagram for the Local rules subtab

Figure 5.5 shows the subtab ”Vendors” in the signature tab which lists available vendors. When the subtab is selected, Cockpit’s spawn function is called to execute the ”ls /var/lib/suricata/update/-sources” command to obtain the list of vendors which is then displayed in a table that can be sorted alphabetically. In the next steps the administrator can:

- Add vendor source
- Remove vendor source
- Enable vendor
- Remove secret key
- Disable vendor
- Fetch vendors
- Apply changes

The above options are executed similarly, in the sense that they use cockpit.spawn to spawn suricata-update with different arguments based on the chosen option. The main differences are with fetch vendors and apply changes. The fetch vendor fetches the index file from OISF, if an index file already exists it will be overwritten by the newly fetched one.

Apply changes will spawn a process with the command *suricata-update* that checks every vendors if they have been enabled or disabled, and if any custom vendors have been added or removed. When the check is done the next step is to download the enabled vendors and unpack their compressed files to the *suricata.rules* file. Once this is done the Suricata service is reloaded in order to make use of the updated *suricata.rules* file. Lastly, the output of *Suricata update* will be returned to the subtab and displayed in a dialogue window. The output will notify the administrator about whether or not the changes were successfully applied.

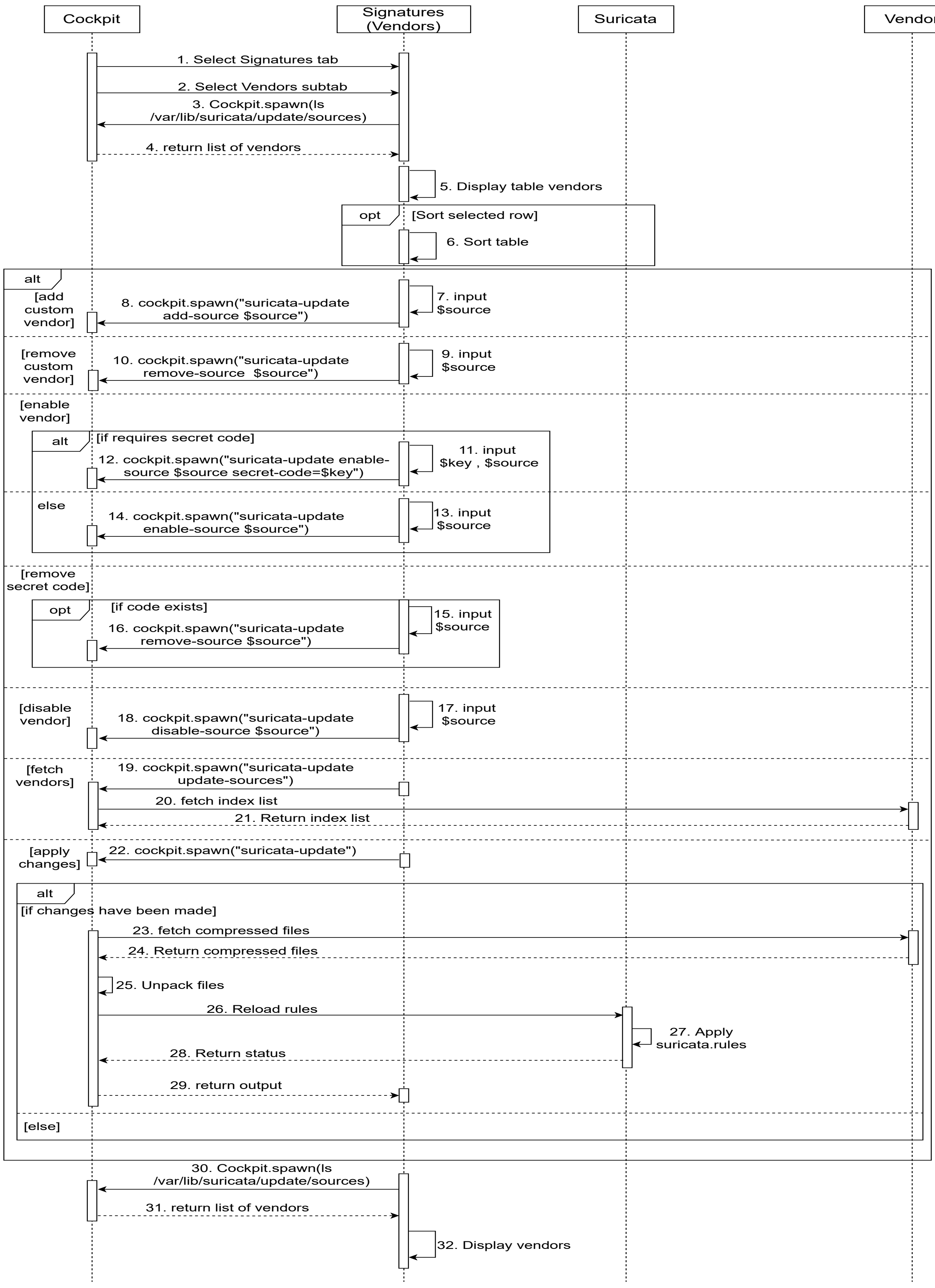

Cockpit
Signatures (Vendors)
Suricata
Vendor
1. Select Signatures tab
2. Select Vendors subtab
3. Cockpit.spawn(ls /var/lib/suricata/update/sources)
4. return list of vendors
5. Display table vendors
opt
[Sort selected row]
6. Sort table
alt
[add custom vendor]
7. input $source
8. cockpit.spawn("suricata-update add-source $source")
[remove custom vendor]
9. input $source
10. cockpit.spawn("suricata-update remove-source $source")
[enable vendor]
alt
[if requires secret code]
11. input $key , $source
12. cockpit.spawn("suricata-update enable-source $source secret-code=$key")
else
13. input $source
14. cockpit.spawn("suricata-update enable-source $source")
[remove secret code]
opt
[if code exists]
15. input $source
16. cockpit.spawn("suricata-update remove-source $source")
[disable vendor]
17. input $source
18. cockpit.spawn("suricata-update disable-source $source")
[fetch vendors]
19. cockpit.spawn("suricata-update update-sources")
20. fetch index list
21. Return index list
[apply changes]
22. cockpit.spawn("suricata-update")
alt
[if changes have been made]
23. fetch compressed files
24. Return compressed files
25. Unpack files
26. Reload rules
27. Apply suricata.rules
28. Return status
29. return output
[else]
30. Cockpit.spawn(ls /var/lib/suricata/update/sources)
31. return list of vendors
32. Display vendors

Figure 5.5: Sequence Diagram for the Vendor subtab

5.3.3 The Logs tab

Figure 5.6 shows the sequence diagram for the module's Logs tab, which is responsible for displaying different log files. The user navigates to the Logs tab and can then choose to display logs for systemd or *stats.log* from the respective subtabs. To read the systemd logs, systemd's journalctl utility is spawned using Cockpit's API [33] for spawning processes. The *stats.log* file is initially read and displayed using Cockpit's API that provides the functionality to read files with *cockpit.read()*. The file is then monitored and read by spawning the tail process. It is also possible to filter the logs from systemd and *stats.log*. In the end the result will be displayed on the web interface.

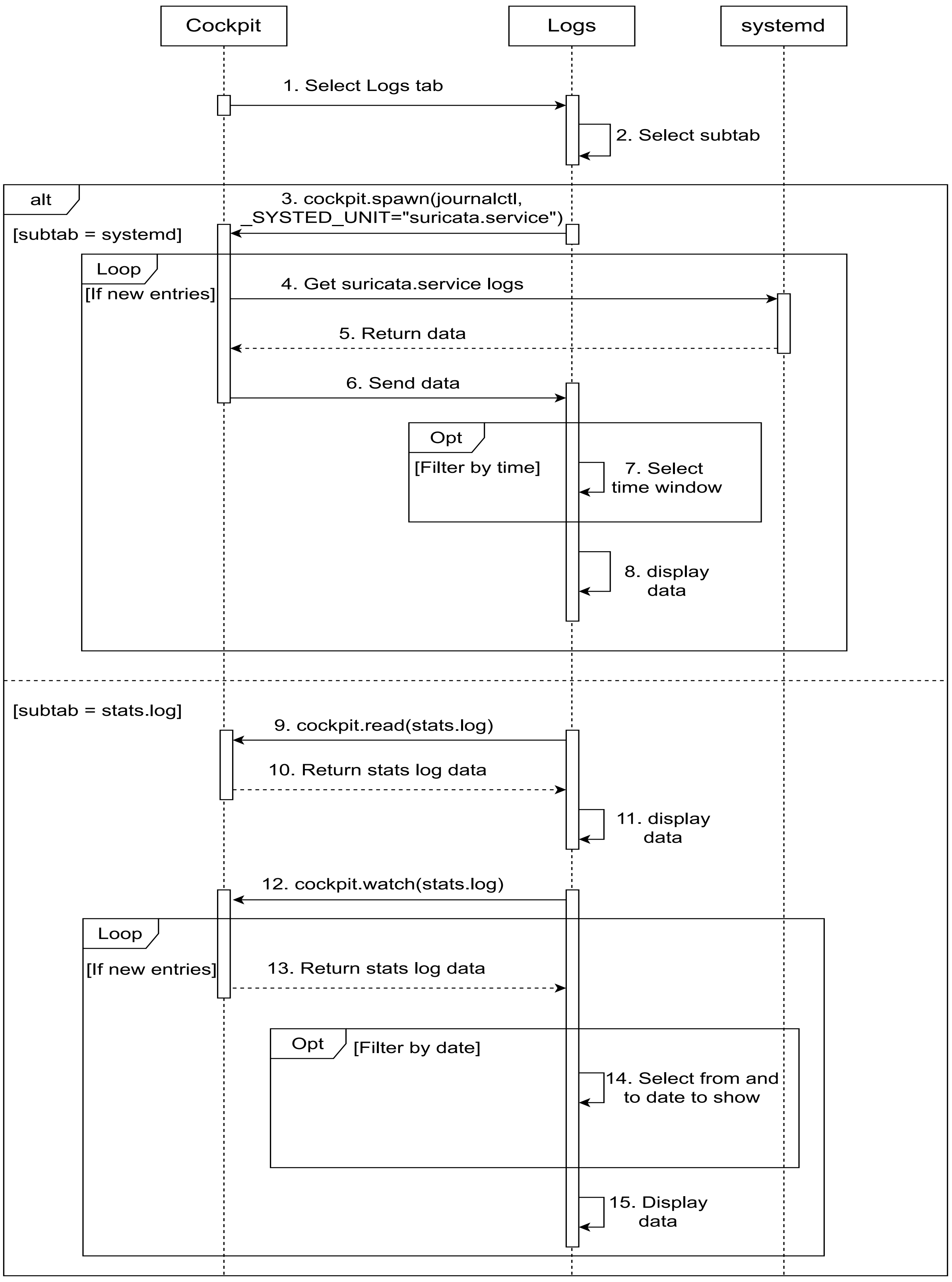


Figure 5.6: Sequence Diagram for the Logs tab

5.3.4 The Alerts tab

Figure 5.7 shows the sequence diagram for the module's Alerts tab. When the user navigates to the Alerts tab, Cockpit reads the *eve.json* file and returns its content. The content is then parsed and every alert is displayed in a table. Recurring alerts is counted and shown in a new table with the amount of times they have been triggered. Both tables can be sorted.

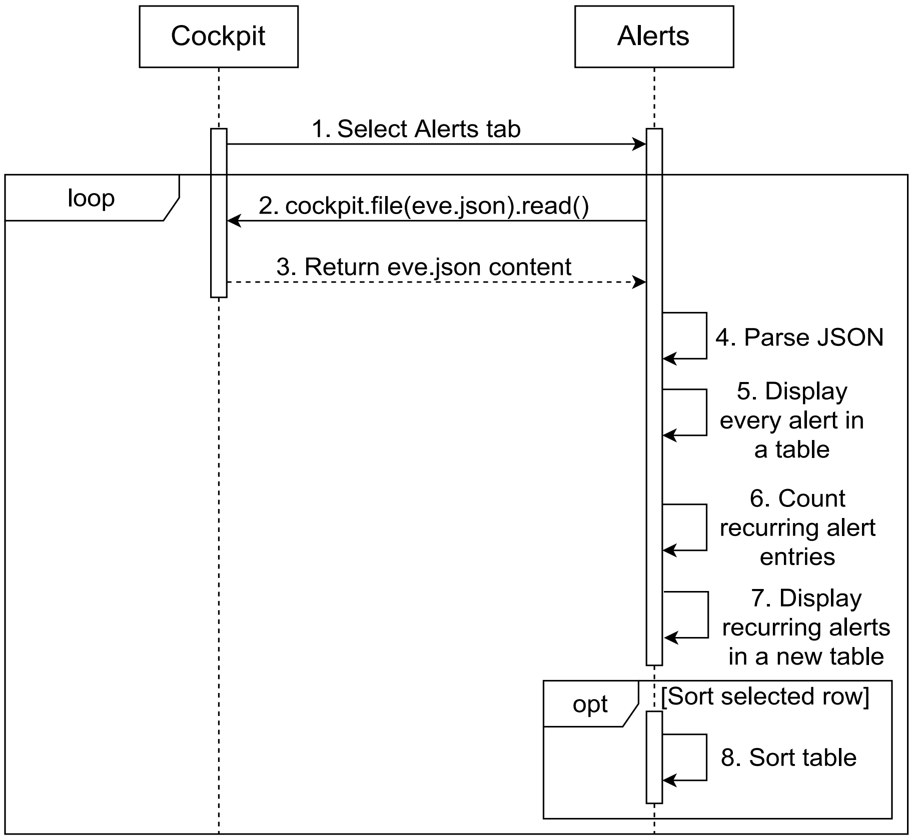

Cockpit
Alerts
1. Select Alerts tab
loop
2. cockpit.file(eve.json).read()
3. Return eve.json content
4. Parse JSON
5. Display every alert in a table
6. Count recurring alert entries
7. Display recurring alerts in a new table
opt
[Sort selected row]
8. Sort table

Figure 5.7: Sequence Diagram for the Alerts tab

5.3.5 The Config tab

Figure 5.8 shows the sequence diagram for the module's Config tab. If selecting either "suricata.yaml" or "update.yaml" subtab, the configuration is copied into an object and the text to an editable window that can be accessed by the "Edit directly" button. When copying is done it will start iterating the object to determine how it should be displayed. If the nested object is an array, it will display the array as a table. If the nested object represent a string it will display an editable form.

If selecting the "config files" subtab the administrator can modify the content of any rule filtering files used by Suricata [25]. At the end the administrator has the option to edit any of the configuration files and save the changes.

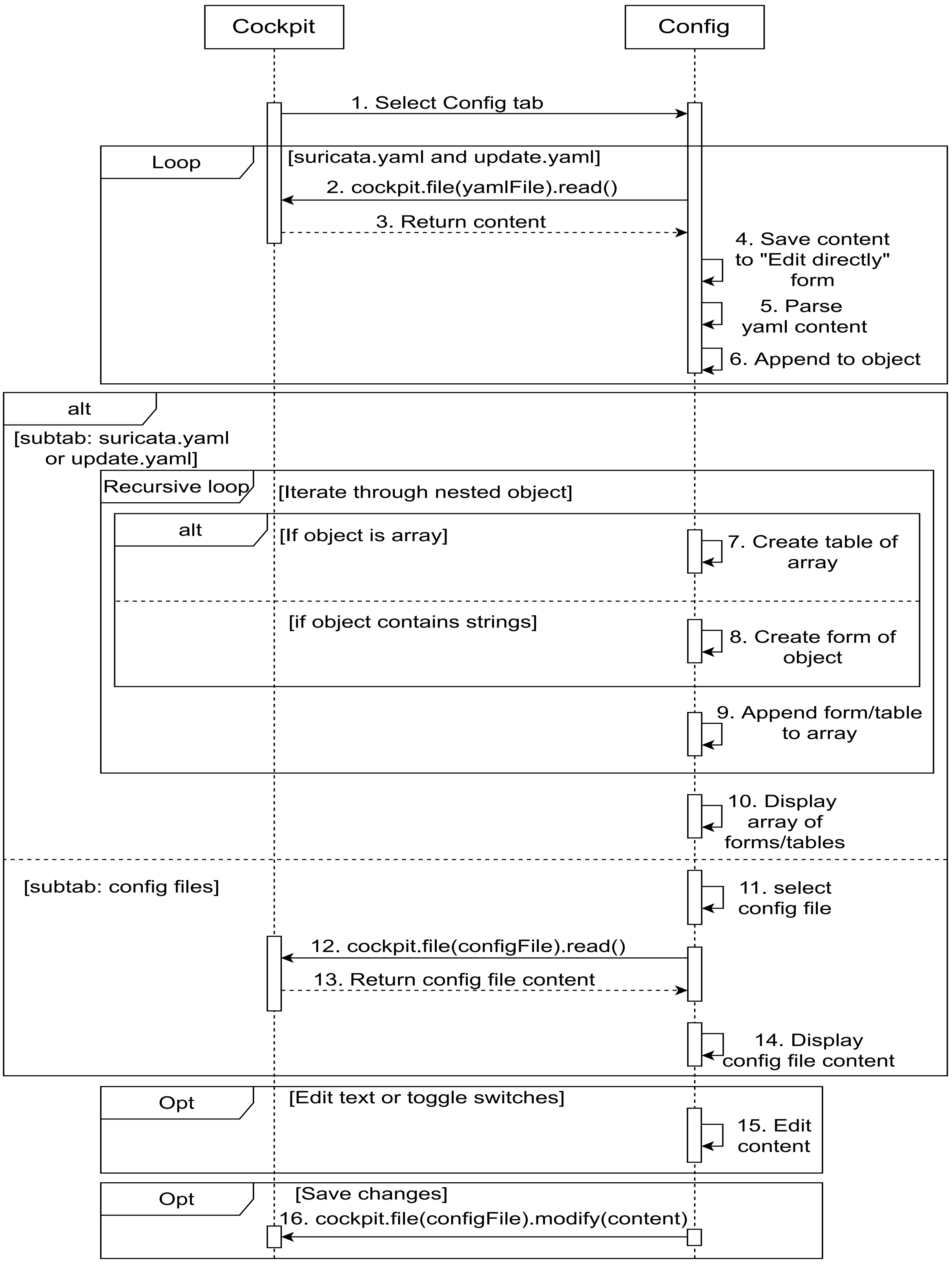

Cockpit
Config
1. Select Config tab
Loop
[suricata.yaml and update.yaml]
2. cockpit.file(yamlFile).read()
3. Return content
4. Save content to "Edit directly" form
5. Parse yaml content
6. Append to object
alt
[subtab: suricata.yaml or update.yaml]
Recursive loop
[Iterate through nested object]
alt
[If object is array]
7. Create table of array
[if object contains strings]
8. Create form of object
9. Append form/table to array
10. Display array of forms/tables
[subtab: config files]
11. select config file
12. cockpit.file(configFile).read()
13. Return config file content
14. Display config file content
Opt
[Edit text or toggle switches]
15. Edit content
Opt
[Save changes]
16. cockpit.file(configFile).modify(content)

Figure 5.8: Sequence Diagram for the Config tab

Chapter 6

Implementation

This chapter describes how the Cockpit module for Suricata IDS was implemented. It starts off by describing important decisions we made regarding the software licence and development environment. Then the graphical user interface is described, followed by key code snippets of the implementation. Finally the chapter ends with the documentation of the module.

6.1 Software License

The Cockpit module for Suricata will be open source and in communication with our client we agreed upon one of the following two licences:

- GNU General Public Licence (GPL):
  The GNU General Public Licence [34] is a license meant to allow users to study, share, run and modify the software freely. Any work based on the original can only be distributed using the same or equivalent license terms. Currently GPL consists of three different versions:
  - GPLv1:
    The first version requires distributors to also publish the human-readable source code along with their binary files under the same licensing terms.
  - GPLv2:
    The second version states that a distributor can use other licenses to publish a work covered by GPL if the license satisfies all of GPL's terms.
  - GPLv3:
    The third version further increased GPL's compatibility with other free software licences, and generally aims to make the rules about GPL-compatible licenses clearer [35]. Additionally changes were done regarding software patents, what defines ”source code”, and to reduce the restriction that hardware can have on software modifications.
- Massachusetts Institute of Technology (MIT):
  The MIT license is a permissive license and is therefore compatible with many copyleft licenses including GPL, due to its focus on few restrictions on reuse. This means MIT licensed software

can be integrated with, or re-licensed as GPL software, but not the other way around [36].

We wanted to ensure that the freedom of our software does not get restricted by the users it gets distributed to, so we opted for GPL over MIT. Coincidentally, Cockpit is licensed under GPLv2 with a clause that says "or any later version" to allow the flexible optional use of either version 2 or 3. Since we use some of Cockpit's code, and we have used Cockpit Starter Kit (which is under the same license), our module must be distributed under the same or equivalent license terms. Our module will therefore be using the GPLv2 license with the "or any later version" clause.

6.2 Development Environment

This section describes the development environment that was used to develop and test our module. It talks about the operating system, module location and different tools we used when developing the module.

Operating System

We decided to develop our module on Ubuntu because it is one of the more commonly used Linux distributions that is supported by both Suricata and Cockpit.

Cockpit module location

Our module is separated into various directories with a combination of HTML, JavaScript and JavaS-cript Object Notation files. The directory name coincides with the name of the module as it is shown on the Cockpit dashboard. For Cockpit to locate the module, they need to be placed in, or a symbolic link needs to be made to, one of three default locations:

- $HOME/.local/share/cockpit/
- /usr/local/share/cockpit/
- /usr/share/cockpit/

Modules in *$HOME/.local/share/cockpit/* directory are only available to the owner of the home directory. While modules in the "/usr/local/share/cockpit/" and "/usr/share/cockpit/" are available to all system users.

Cockpit Starter Kit

A production ready project has a lot of requirements, such as using linting to analyze the code for errors, a modern framework like React and a build system. Cockpit provides a starter kit [37], which enabled us start coding straight away instead of learning the details of all these requirements first. The starter kit also provides a simple example project as starting point to build on. Martin Pitt has written an article on this and explains how to get started [1].

[1]https://cockpit-project.org/blog/cockpit-starter-kit.html

npm

Node Package Manager (npm) is a package manager tool for JavaScript, which is bundled with NodeJs. Our module in itself does not require NodeJs, but npm is dependent on NodeJs when executed. The tool uses *package.json* as its configuration file which includes the needed packages and additional custom scripts. With custom scripts we have it set up to build with webpack and other convenient entries like executing Eslint linter seen in Code listing 6.1.

Code listing 6.1: Snippet of package.json demonstration custom scripts

```
"scripts": {
  "watch": "webpack --watch --progress",
  "build": "webpack",
  "eslint": "eslint --ext .js --ext .jsx src/",
  "eslint:fix": "eslint --fix --ext .js --ext .jsx src/"
},
```

Webpack

Webpack [38] is an open source JavaScript module bundler. When webpack processes an application it internally builds a dependency graph which maps every module the project needs and generates one or more bundles, which in turn allows us to use a modular approach. By using webpack configuration we can specify plugins that can help optimize and implement debugging during the build of our module.

GNU make

GNU make [39] is a utility that run tasks defined in a file named *Makefile*. In our case running the *make* command will compile our code with webpack. To automatically compile webpack on code change, the command ”make watch” can be used. These changes will only be visible to the user that ran the command because the webpack generated output has a symbolic link to *$HOME/.local/share/cockpit* directory. To make the changes available to all system users, the command ”make install” needs to be executed. The command will copy the output to */usr/share/cockpit* directory making the module available system-wide.

ESlint

ESlint [40] is used to automatically analyze code for problems, e.g., if the coding style is breaking the defined rules. It helps us make our code more consistent and avoid bugs. It is run at every build using webpack loader, but can also be run manually with the command ”npm run eslint”. If ESlint finds rule violations, they can be fixed using the command ”npm run eslint:fix”. The rule configuration can be found in the *.eslintrc.json* file.

Packages

Our module uses packages that are managed by npm. To see all the packages we use for our module, check out the *package.json* file in the github repository [2].

[2]https://github.com/Sudland/cockpit-suricata/blob/master/package.json

6.2.1 Libraries

React
React [41] is an open source, front end, JavaScript library for building user interfaces. React allows us to build reusable components that can be combined to create complex user interfaces.

PatternFly
PatternFly [42] is an open source design system created to enable consistency and usability by providing clear standards, guidance, and tools that help designers and developers work together more efficiently. Cockpit's current interface is built using PatternFly. By using PatternFly, our module blends in well with the rest of Cockpit's layout.

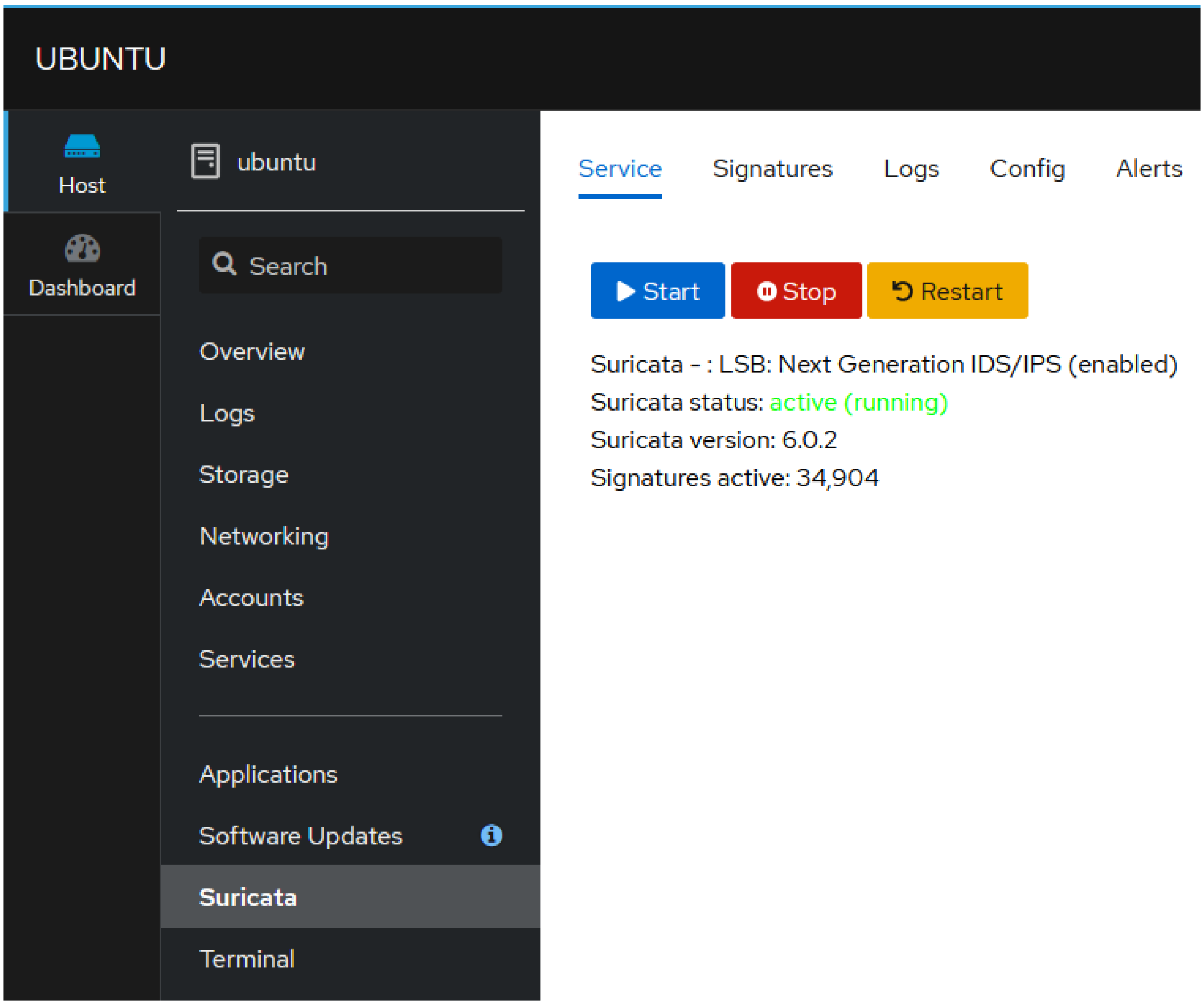
Puppeteer
Puppeteer [43] is a Node library which provides an API to control Chrome or Chromium. Puppeteer runs in the background and can be programmed to follow instructions. We used Puppeteer to create a testing environment for doing things we would have to do manually in a browser, e.g., log in to Cockpit, navigate through different tabs and so forth.

6.3 Graphical User Interface of Our Module

In this section we will introduce the GUI that we have made for the module. We decided to make the GUI for the module as close as possible to the design of Cockpit. We designed the interface with the help of PatternFly since this is what Cockpit uses [44].

6.3.1 The Service tab

Figure 6.1 shows the Service tab, which is the tab the administrator will be presented with when first accessing the module on Cockpit. On this tab the administrator can start, stop and restart Suricata by pressing the corresponding buttons. Below the buttons is information about the current status of the Suricata IDS service as well as information about which version is in use and how many signatures are active.

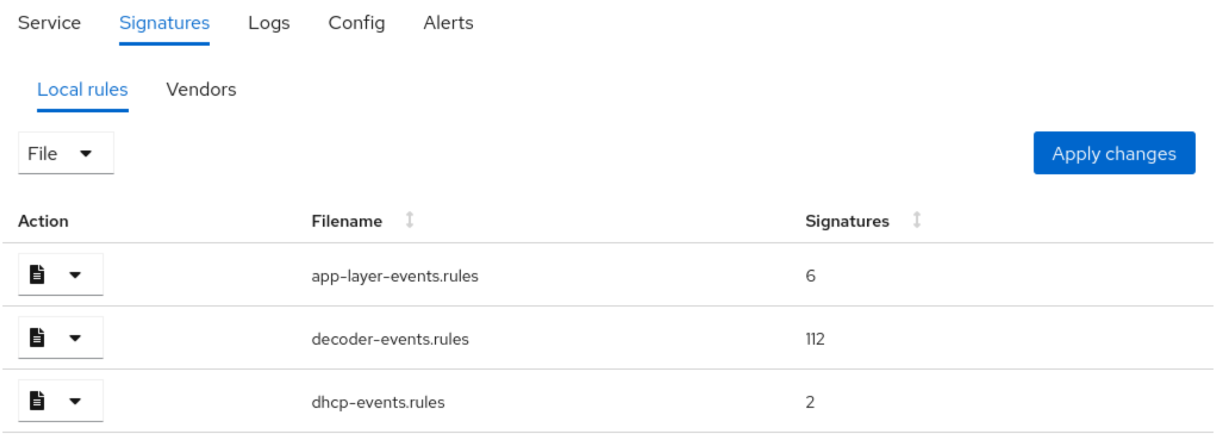


Suricata - : LSB: Next Generation IDS/IPS (enabled)
Suricata status: active (running)
Suricata version: 6.0.2
Signatures active: 34,904

Figure 6.1: The front page of the module

6.3.2 The Signatures tab

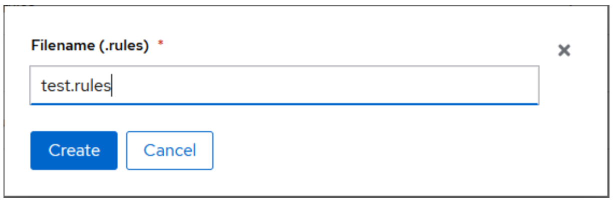
Figure 6.2 shows the signatures tab consisting of the ”Local rules” and ”Vendors” subtabs. On the ”Local rules” subtab the administrator can create, upload, edit and delete rule sets used by Suricata.



Action	Filename	Signatures
	app-layer-events.rules	6
	decoder-events.rules	112
	dhcp-events.rules	2

Figure 6.2: The Local rules subtab on the Signatures tab

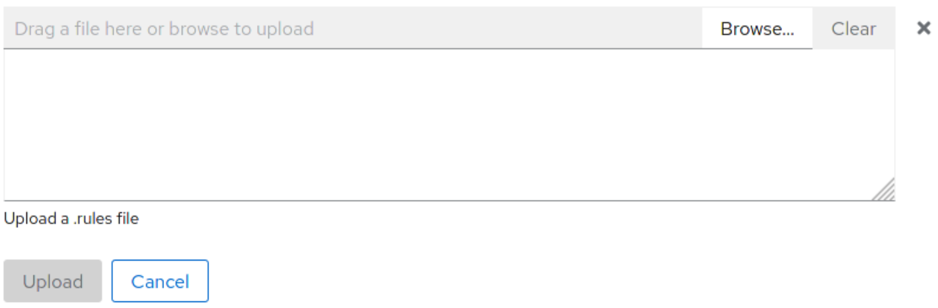
In order to create a rule set the administrator can press the ”File” drop-down menu and select ”New file”. If the administrator wants to create a new rule set, he or she needs to enter a name for the file followed by the ”.rules” extension and press the ”Create” button which can be seen in Figure 6.3.


Filename (.rules) *

test.rules

Figure 6.3: The dialogue window for the create file operation

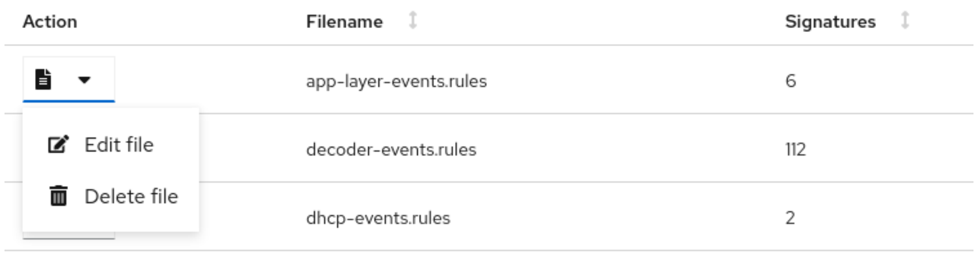
To upload a rule set the administrator selects the ”Upload file” from the ”File” drop-down menu and can either drag and drop or browse the file system to add an existing file, as shown in Figure 6.4. The file name must end with ”.rules” to be able to upload the file.



Upload a .rules file

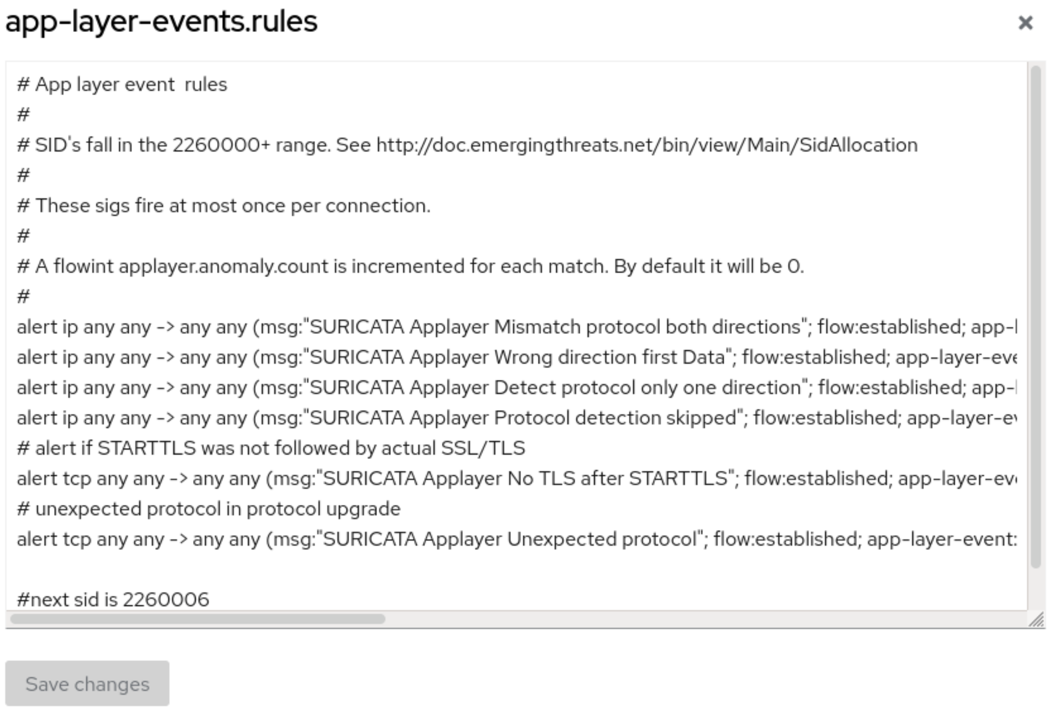
Figure 6.4: The dialogue window where an administrator can upload an existing file

To delete or edit the content of the file the administrator can press the ”Action” drop-down menu shown in Figure 6.5 and select "Delete file" or "Edit file". The "Edit file" action will open up the file in an editable text panel as shown in Figure 6.6. The "Delete file" action prompts the administrator with a dialogue window asking if they really want to delete the file.



Action	Filename	Signatures
	app-layer-events.rules	6
	decoder-events.rules	112
	dhcp-events.rules	2

Figure 6.5: The actions an administrator can perform on a file



app-layer-events.rules

```
# App layer event  rules
#
# SID's fall in the 2260000+ range. See http://doc.emergingthreats.net/bin/view/Main/SidAllocation
#
# These sigs fire at most once per connection.
#
# A flowint applayer.anomaly.count is incremented for each match. By default it will be 0.
#
alert ip any any -> any any (msg:"SURICATA Applayer Mismatch protocol both directions"; flow:established; app-l
alert ip any any -> any any (msg:"SURICATA Applayer Wrong direction first Data"; flow:established; app-layer-eve
alert ip any any -> any any (msg:"SURICATA Applayer Detect protocol only one direction"; flow:established; app-l
alert ip any any -> any any (msg:"SURICATA Applayer Protocol detection skipped"; flow:established; app-layer-ev
# alert if STARTTLS was not followed by actual SSL/TLS
alert tcp any any -> any any (msg:"SURICATA Applayer No TLS after STARTTLS"; flow:established; app-layer-ev
# unexpected protocol in protocol upgrade
alert tcp any any -> any any (msg:"SURICATA Applayer Unexpected protocol"; flow:established; app-layer-event:

#next sid is 2260006
```

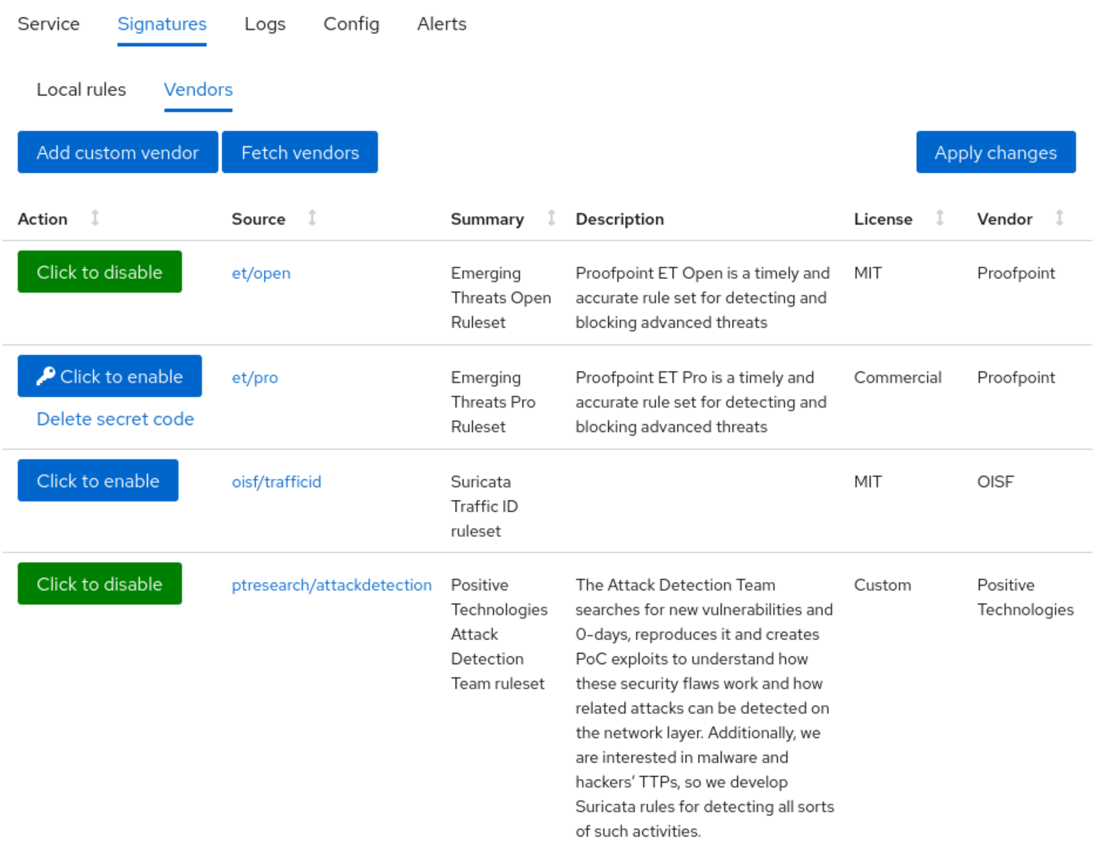
Figure 6.6: The text box window where an administrator can edit a file

Figure 6.7 shows the ”Vendors” subtab where the administrator can ”Add custom vendor”, ”Fetch vendors” and enable or disable signature sources.

If the module can not find ”/var/lib/suricata/update/cache/index.yaml”, an alert is given in the top right corner telling the administrator that the *index.yaml* file is missing, and that they can press fetch vendors to fix it. When the administrator presses the fetch vendors button, the module will download the rule sets from https://www.openinfosecfoundation.org/rules/index.yaml

The administrator has the option to enable and disable rule sets by clicking on a rule set’s ”Click to enable” or ”Click to disable” button, depending on that rule set’s current state. ”Click to disable” buttons are green to indicate that those rule sets are currently enabled. ”Click to enable” buttons are blue and those rule sets are currently disabled.

For any changes in the Signatures tab to take effect the ”Apply changes” button has to be pressed. Once the button is pressed, the Suricata service is reloaded in order to make use of the new changes. A dialogue window showing whether the changes were successfully applied or not is then shown to the administrator.



Action	Source	Summary	Description	License	Vendor
Click to disable	et/open	Emerging Threats Open Ruleset	Proofpoint ET Open is a timely and accurate rule set for detecting and blocking advanced threats	MIT	Proofpoint
Click to enable Delete secret code	et/pro	Emerging Threats Pro Ruleset	Proofpoint ET Pro is a timely and accurate rule set for detecting and blocking advanced threats	Commercial	Proofpoint
Click to enable	oisf/trafficid	Suricata Traffic ID ruleset		MIT	OISF
Click to disable	ptresearch/attackdetection	Positive Technologies Attack Detection Team ruleset	The Attack Detection Team searches for new vulnerabilities and 0-days, reproduces it and creates PoC exploits to understand how these security flaws work and how related attacks can be detected on the network layer. Additionally, we are interested in malware and hackers’ TTPs, so we develop Suricata rules for detecting all sorts of such activities.	Custom	Positive Technologies

Figure 6.7: Shows the Vendors subtab on the Signatures tab

Figure 6.8 shows what happens when the ”Add custom vendor” button is pressed. The administrator gives the custom vendor a name and provides the URL for the vendor before pressing ”Submit”. The source is then added to the list of vendors. The source URL must be a valid URL to be able to submit.

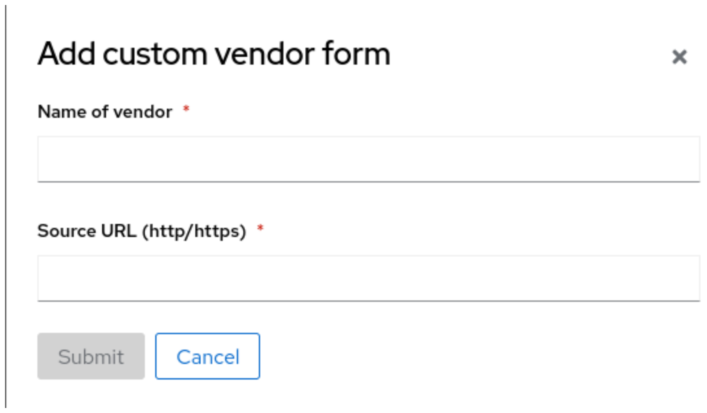


Figure 6.8: The dialogue window where an administrator can add a new vendor source to Suricata

A vendor might require a secret code to enable it. In that case a key will be shown on the ”Click to enable” button. When the button is pressed the administrator will be prompted to enter the secret code as shown in Figure 6.9. Vendors with required secret code that previously was enabled will store the key and give the administrator the option to delete the stored secret key.

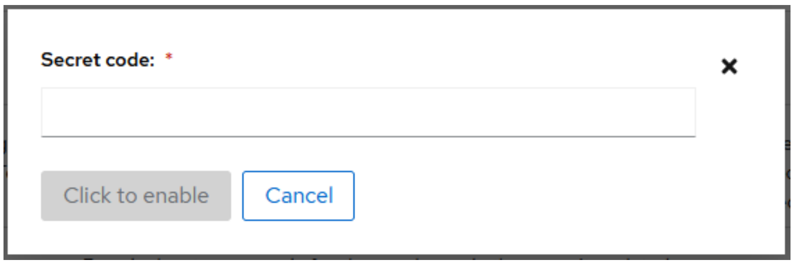
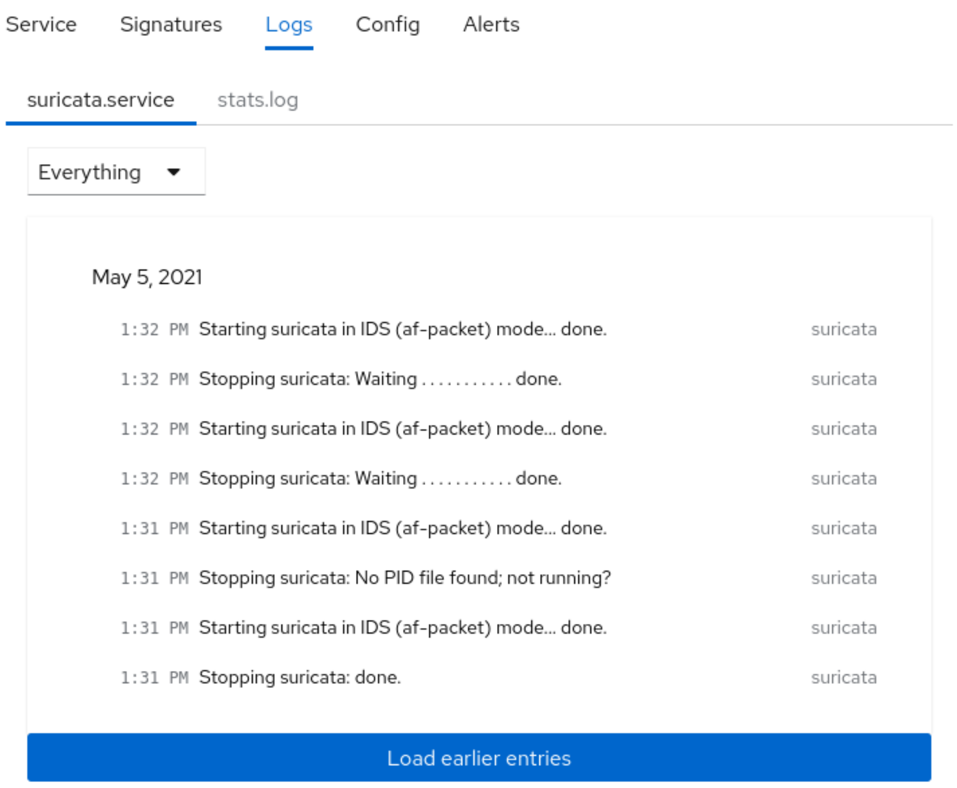


Figure 6.9: How to add a secret code

6.3.3 Logs

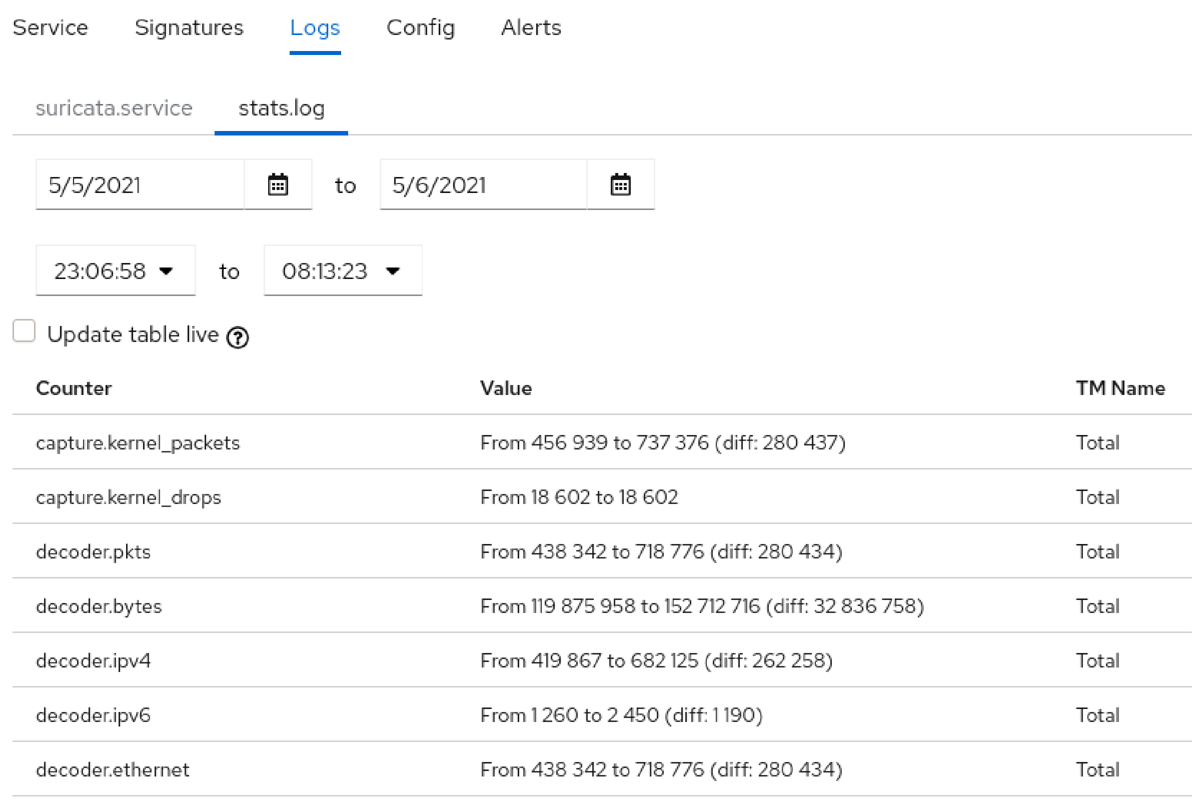
Figure 6.10 shows the ”suricata.service” default subtab under the Logs tab. This subtab shows the systemd logs for the Suricata service. The administrator can choose which time period to show logs by selecting from the drop-down menu seen in Figure 6.10. The choices are ”Everything”, ”Current boot”, ”Previous boot”, ”Last 24 hours” and ”Last 7 days”.



```
May 5, 2021
1:32 PM Starting suricata in IDS (af-packet) mode... done.        suricata
1:32 PM Stopping suricata: Waiting ........... done.              suricata
1:32 PM Starting suricata in IDS (af-packet) mode... done.        suricata
1:32 PM Stopping suricata: Waiting .......... done.               suricata
1:31 PM Starting suricata in IDS (af-packet) mode... done.        suricata
1:31 PM Stopping suricata: No PID file found; not running?        suricata
1:31 PM Starting suricata in IDS (af-packet) mode... done.        suricata
1:31 PM Stopping suricata: done.                                  suricata
```

Figure 6.10: Shows the suricata.service subtab on the Logs tab

Figure 6.11 shows the ”stats.log” subtab under the Logs tab. This subtab shows the content of the *stats.log* file in a table. The *stats.log* file grows with time, as new entries are appended to the file at given intervals. In order to be able to see earlier entries, or see entries in a specific time period, the administrator can sort the table by using the from and to *date* fields. Below the from and to date fields are from and to *time* fields to compare the difference between the given times. This difference can be seen in parenthesis in the ”Value” column. The administrator can also have the table updating in real-time by checking the ”Update table live” box.

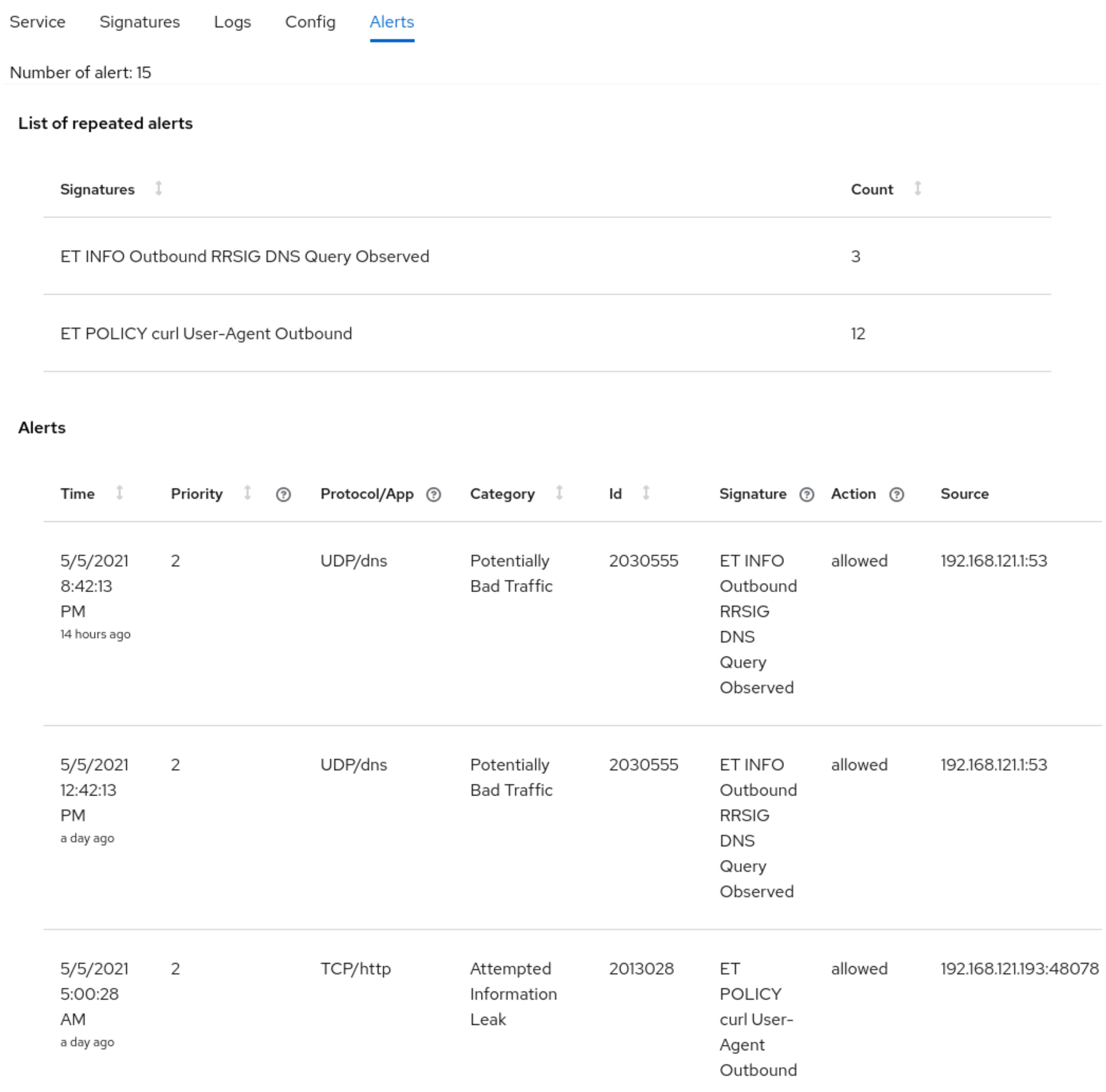


Counter	Value	TM Name
capture.kernel_packets	From 456 939 to 737 376 (diff: 280 437)	Total
capture.kernel_drops	From 18 602 to 18 602	Total
decoder.pkts	From 438 342 to 718 776 (diff: 280 434)	Total
decoder.bytes	From 119 875 958 to 152 712 716 (diff: 32 836 758)	Total
decoder.ipv4	From 419 867 to 682 125 (diff: 262 258)	Total
decoder.ipv6	From 1 260 to 2 450 (diff: 1 190)	Total
decoder.ethernet	From 438 342 to 718 776 (diff: 280 434)	Total

Figure 6.11: Shows the stats.log subtab on the Logs tab

6.3.4 Alerts

Figure 6.12 shows the module's ”Alerts” tab. This tab first displays how many alerts have been triggered in total. Below that is a table of which alerts have been triggered, and a count for how many times they have been triggered. Lastly it shows all alerts triggered by the Suricata IDS service in the bottom table. The table is automatically updated when a new alert is triggered, but in order to not overload the browser, only the latest fifty alerts are shown in the table. There is a ”Click to expand” button at the bottom of the table which shows the next fifty alerts when clicked.



Number of alert: 15

List of repeated alerts

Signatures	Count
ET INFO Outbound RRSIG DNS Query Observed	3
ET POLICY curl User-Agent Outbound	12

Alerts

Time	Priority	Protocol/App	Category	Id	Signature	Action	Source
5/5/2021 8:42:13 PM 14 hours ago	2	UDP/dns	Potentially Bad Traffic	2030555	ET INFO Outbound RRSIG DNS Query Observed	allowed	192.168.121.1:53
5/5/2021 12:42:13 PM a day ago	2	UDP/dns	Potentially Bad Traffic	2030555	ET INFO Outbound RRSIG DNS Query Observed	allowed	192.168.121.1:53
5/5/2021 5:00:28 AM a day ago	2	TCP/http	Attempted Information Leak	2013028	ET POLICY curl User-Agent Outbound	allowed	192.168.121.193:48078

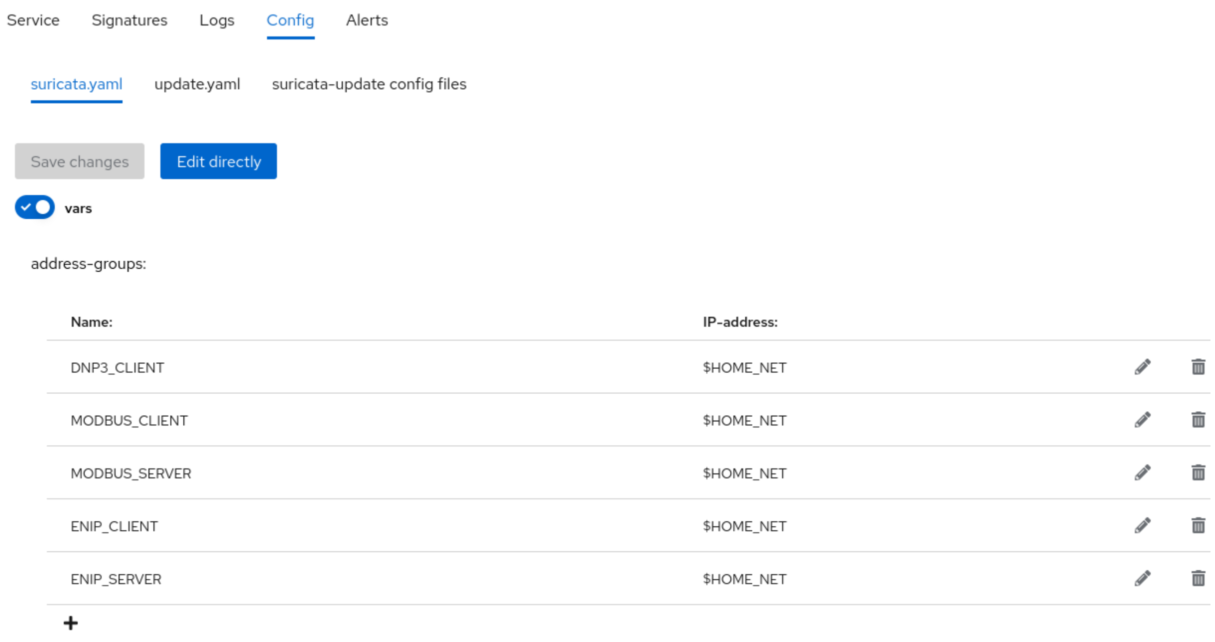
Figure 6.12: Shows the Alerts tab

6.3.5 Config

Figure 6.13 shows the ”Config” tab consisting of the ”suricata.update”, ”update.yaml” and ”suricata-update config files” subtabs.

By default the ”suricata.yaml” subtab is selected. In this subtab the administrator can make changes to Suricata’s configuration file called *suricata.yaml*. This file has many options that can be set and changed. Our module allows the administrator to edit this file directly by clicking the ”Edit directly” button or via a more user friendly interface as seen in the figure.

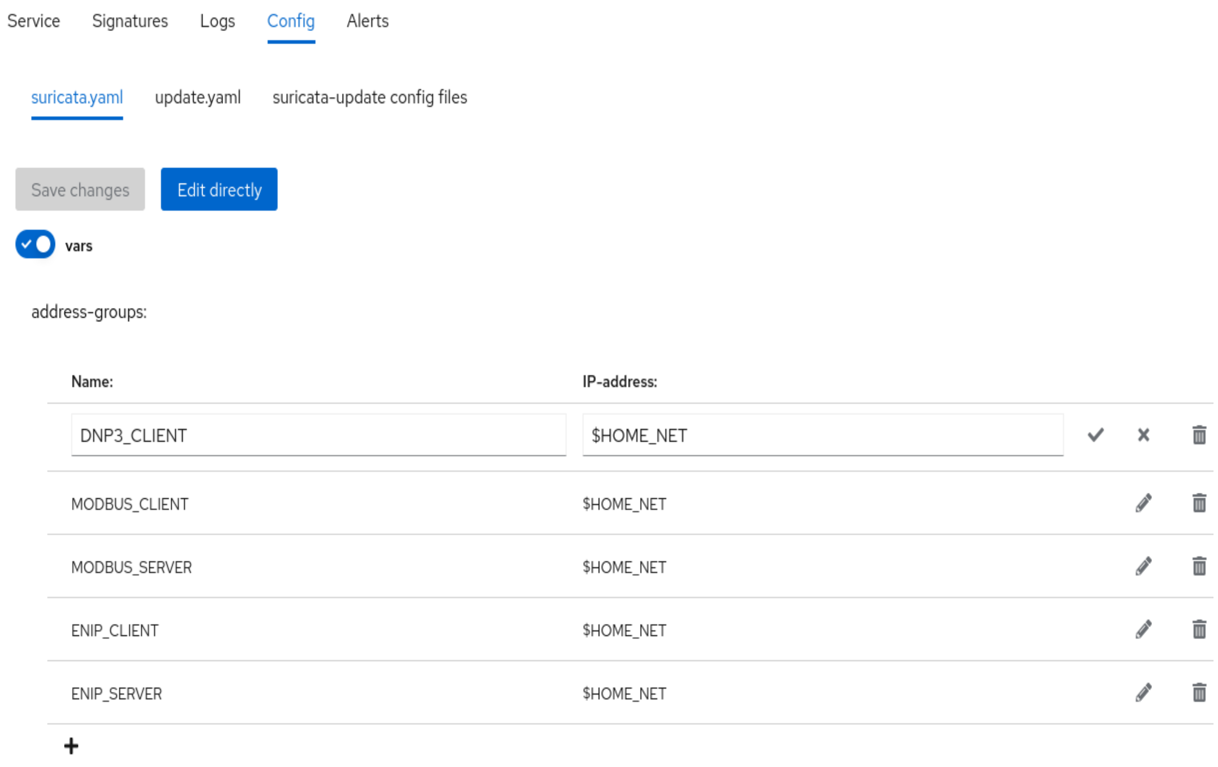
One of these options is *address-groups*. The administrator can use the *vars* switch to enable or disable the option. Enabling and disabling the file is equivalent to commenting and uncommenting the option in the *suricata.yaml* file. The administrator can edit or delete the existing entries to the address-groups by clicking on the respective pen or trash symbols on the far right. New entries can also be added using the plus symbol below the last entry.



Name:	IP-address:
DNP3_CLIENT	$HOME_NET
MODBUS_CLIENT	$HOME_NET
MODBUS_SERVER	$HOME_NET
ENIP_CLIENT	$HOME_NET
ENIP_SERVER	$HOME_NET

Figure 6.13: Shows the Config tab

When the administrator clicks on the pen symbol to edit an existing field, the administrator is allowed to change the fields of that entry. Figure 6.14 shows how this looks for the first entry with the name "DNP3_CLIENT". The changes can be confirmed by clicking the check mark symbol or discarded by clicking the cross symbol. However, for the changes to be saved to the file, the administrator must click the "Save changes" button. Note that the "Save changes" button will change color from grey to blue and be clickable only when changes are made.

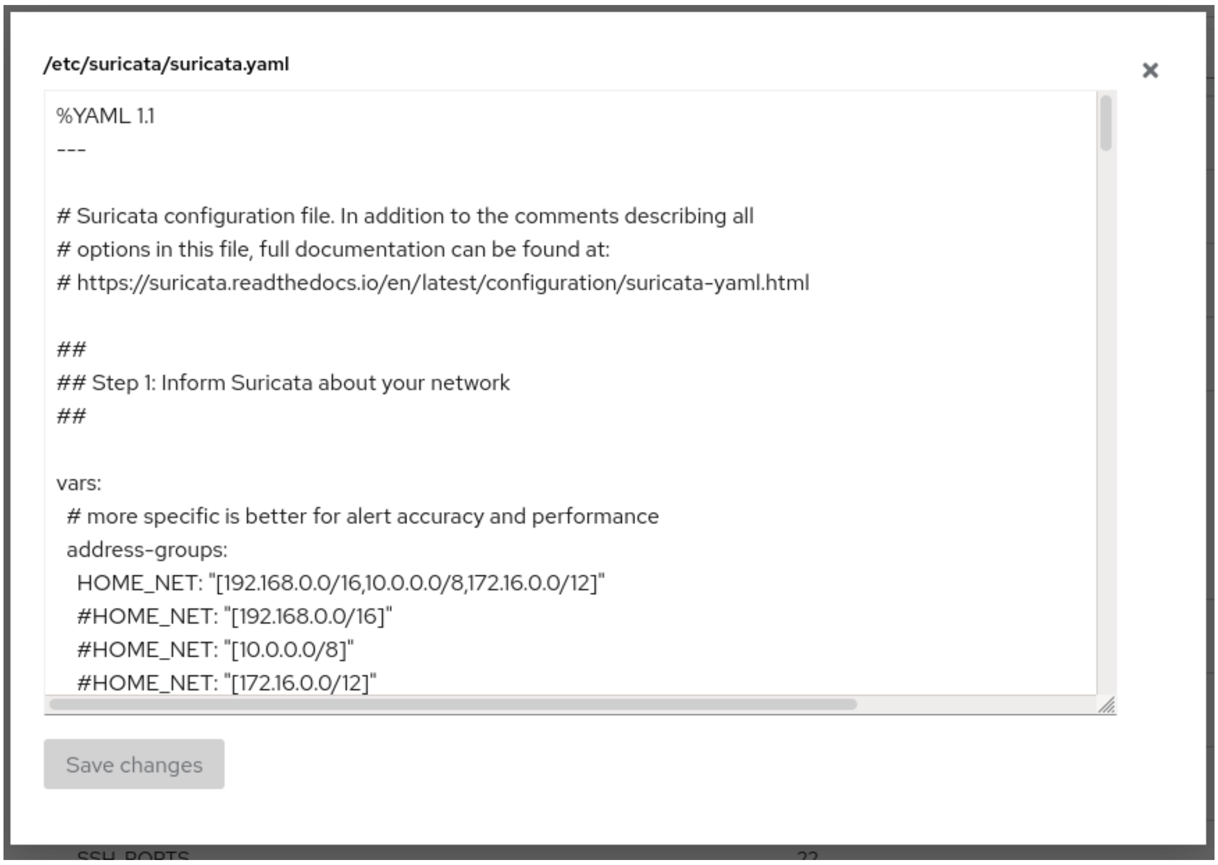


address-groups:

Name:	IP-address:
DNP3_CLIENT	$HOME_NET
MODBUS_CLIENT	$HOME_NET
MODBUS_SERVER	$HOME_NET
ENIP_CLIENT	$HOME_NET
ENIP_SERVER	$HOME_NET

Figure 6.14: Edit cell in a table

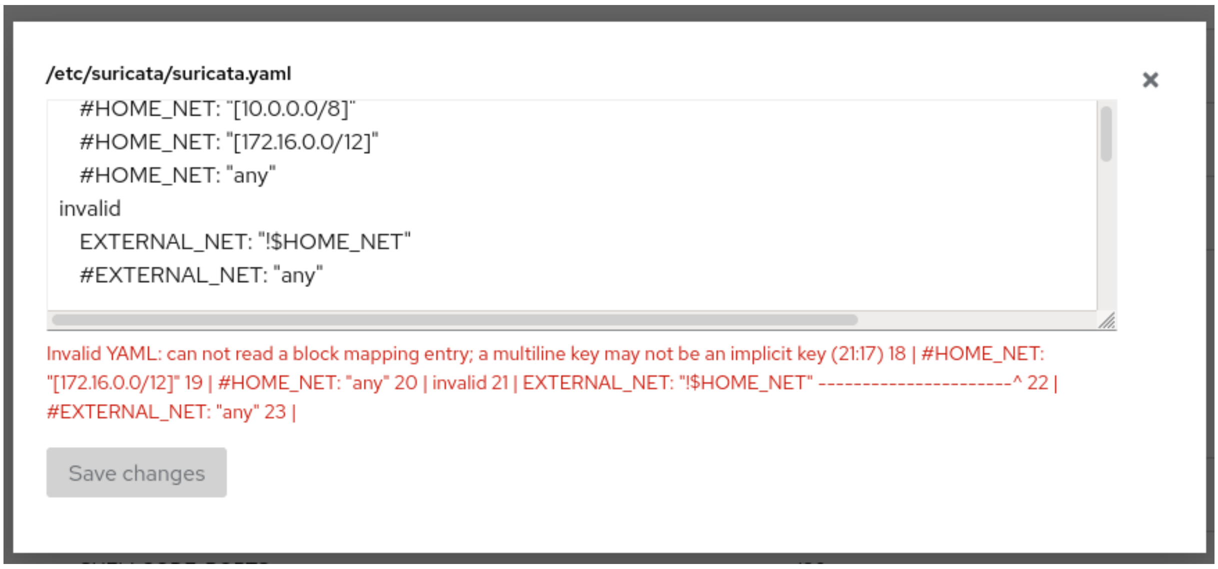
If the administrator clicks on the "Edit directly" button in Figure 6.13, the administrator will be allowed to edit the file directly through the text box shown in Figure 6.15. When the input is invalid YAML syntax the administrator will be warned and also be prevented from saving changes with wrong syntax as shown in Figure 6.16. Therefore ”Save changes” button will only be clickable when changes with valid YAML syntax have been made as seen in Figure 6.17.



/etc/suricata/suricata.yaml

```
%YAML 1.1
---

# Suricata configuration file. In addition to the comments describing all
# options in this file, full documentation can be found at:
# https://suricata.readthedocs.io/en/latest/configuration/suricata-yaml.html

##
## Step 1: Inform Suricata about your network
##

vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
    #HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
```

Figure 6.15: Edit the file directly

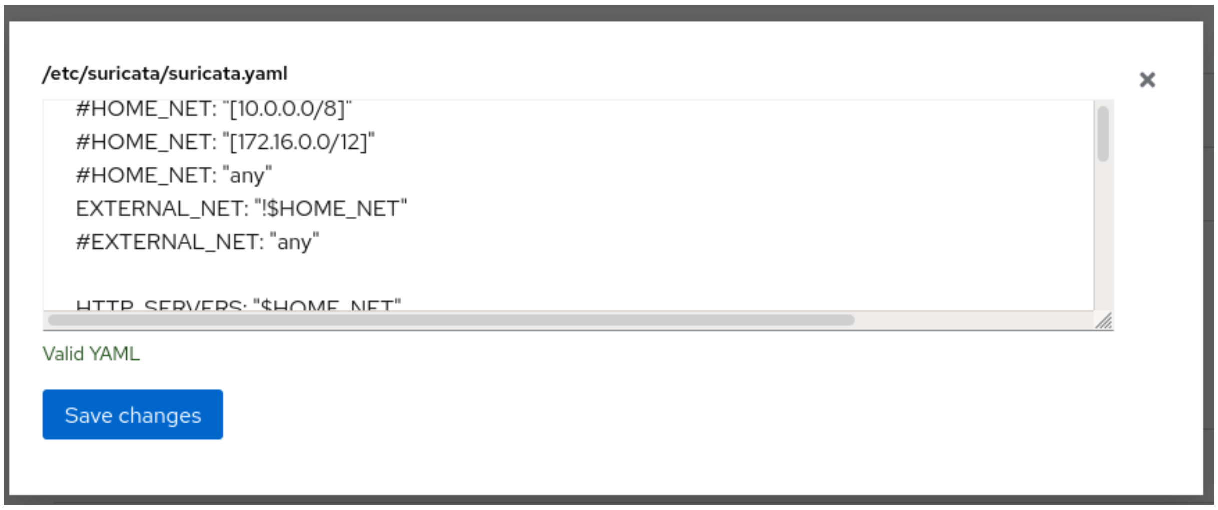


/etc/suricata/suricata.yaml

```
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
    #HOME_NET: "any"
invalid
    EXTERNAL_NET: "!$HOME_NET"
    #EXTERNAL_NET: "any"
```

Invalid YAML: can not read a block mapping entry; a multiline key may not be an implicit key (21:17) 18 | #HOME_NET: "[172.16.0.0/12]" 19 | #HOME_NET: "any" 20 | invalid 21 | EXTERNAL_NET: "!$HOME_NET" ----------------------^ 22 | #EXTERNAL_NET: "any" 23 |

Figure 6.16: Message if invalid syntax inputted

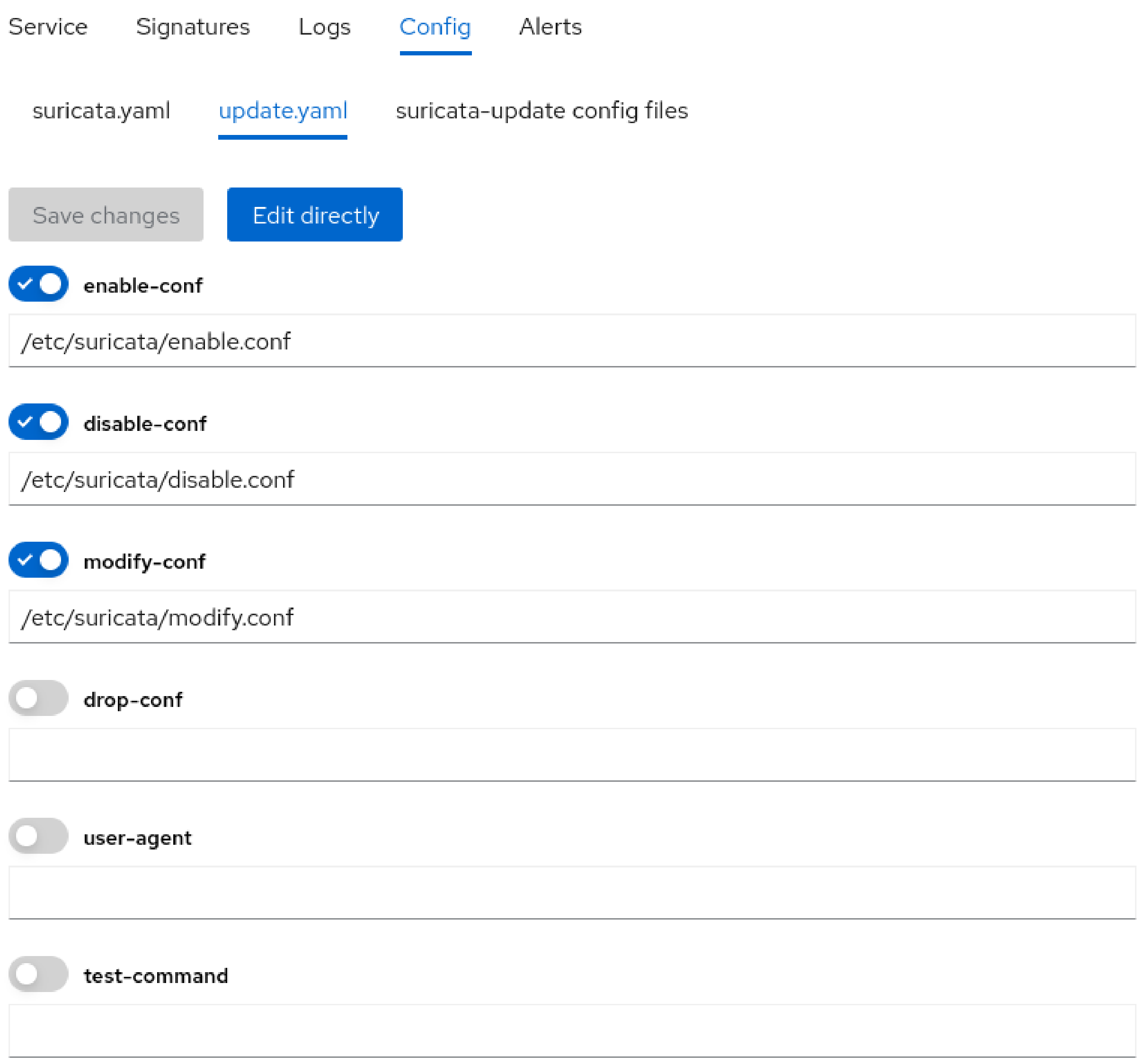

/etc/suricata/suricata.yaml

```
#HOME_NET: "[10.0.0.0/8]"
#HOME_NET: "[172.16.0.0/12]"
#HOME_NET: "any"
EXTERNAL_NET: "!$HOME_NET"
#EXTERNAL_NET: "any"

HTTP_SERVERS: "$HOME_NET"
```

Valid YAML

Figure 6.17: Message if valid syntax inputted

Figure 6.18 shows the ”update.yaml” subtab, which is responsible for making changes to the ”update.yaml” file. The administrator can enable and disable the configuration files *update.yaml* refers to by turning the switch on and off. Other possibilities are to change the path to the different ”.conf” files or edit the *update.yaml* file directly, similar to *suricata.yaml*, as can be seen in Figure 6.15.



✓ enable-conf: /etc/suricata/enable.conf

✓ disable-conf: /etc/suricata/disable.conf

✓ modify-conf: /etc/suricata/modify.conf

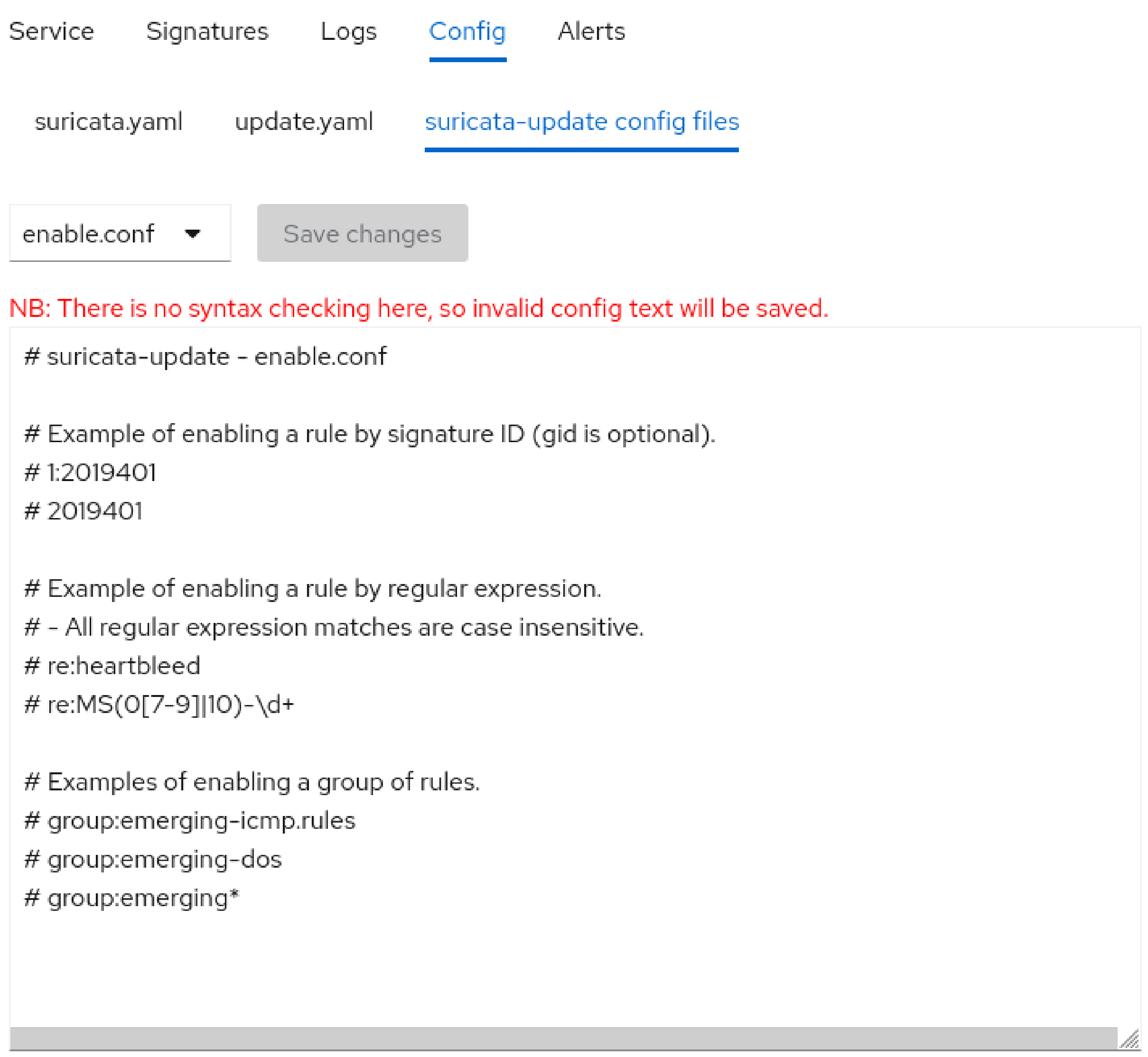
drop-conf:

user-agent:

test-command:

Figure 6.18: Shows the Update.yaml subtab

Figure 6.19 shows the ”suricata-update config files” subtab. This is used by to edit configuration file rules used by ”suricata-update”. This however does not support syntax checking and can be seen stated in the figure.

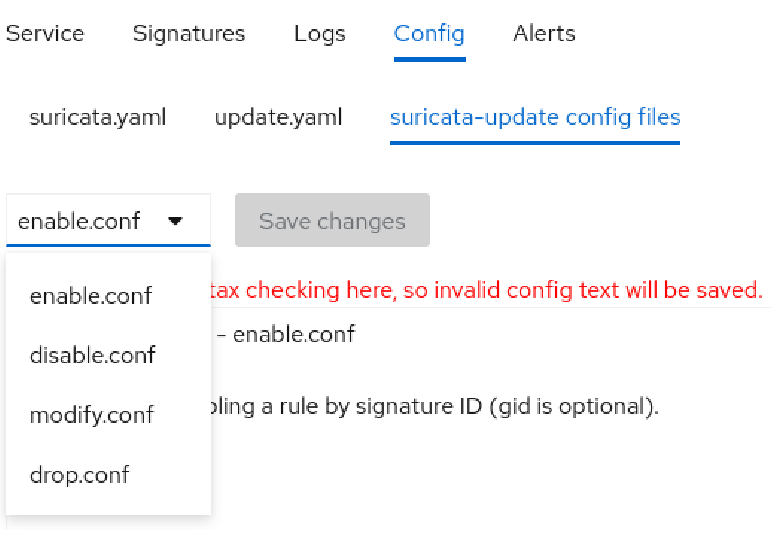


NB: There is no syntax checking here, so invalid config text will be saved.

```
# suricata-update - enable.conf

# Example of enabling a rule by signature ID (gid is optional).
# 1:2019401
# 2019401

# Example of enabling a rule by regular expression.
# - All regular expression matches are case insensitive.
# re:heartbleed
# re:MS(0[7-9]|10)-\d+

# Examples of enabling a group of rules.
# group:emerging-icmp.rules
# group:emerging-dos
# group:emerging*
```

Figure 6.19: suricata-update configuration files

Figure 6.20 shows the drop-down menu where the administrator can select which file to edit. The user interface for editing the different files in the menu is identical.



tax checking here, so invalid config text will be saved.

- enable.conf

oling a rule by signature ID (gid is optional).

Figure 6.20: Drop-down menu of configuration files

6.4 Code

Read and monitor Suricata files In order for us to get the data to be displayed in the module we needed some way to read data that Suricata uses and logs that are generated by Suricata.

Our solution to read the necessary files was to use Cockpit's file API *read()*. The *read()* function can read the file and return the file's content back to our module. We also have the need to make it read continuously to keep track of live changes in some specific files such as *eve.json* and *stats.log* generated by Suricata.

We tried using Cockpit's file API *watch()*. *watch()* works by monitoring a file for any changes and sends a callback with the file's content and can be used to input new data on our module. However this solution started to cause problems. Problems being that in a large network environment Suricata can generate thousands of alerts to *eve.json* every second, making *watch()* keep on sending the whole file's content repeatedly on every change. This can cause severe network bandwidth usage for the client and the server, which in turn will lead to the module becoming unresponsive for the client.

A better way to monitor Suricata files was to use another API called *process* (this will be explained more thoroughly in section 6.4.2). By doing this we can spawn built-in applications in Linux that are more suited for monitoring files. The application we use for this is called **tail** and can also monitor the file, but in this case it can be told to only return newly written data sparing both the client and the server for excessive bandwidth usage.

Code listing 6.2 shows how the *cockpit.file.read()* is implemented in *alerts.jsx*.

Code listing 6.2: *cockpit.file.read()*

```
cockpit
  .file(this.logFile, { max_read_size: 1024 * 1024 * 1024, superuser: 'try' })
  .read()
  .then((content) => {
    if (!content.includes('"event_type":"alert"')) return;
    const { isNew, rows, nrOfAlerts } = this.state;
    const newRows = rows['signature-table'];
    const newRowsCount = [];
    const jsonArray = content.split('\n').filter((x) => x.includes('"event_type":"alert"'));

    if (!isNew) {
      for (let i = 0; i < jsonArray.length; i += 1) {
        let entry;
        try {
          entry = this.pushAlert(JSON.parse(jsonArray[i]));
        } catch {
          console.log(jsonArray[i]);
        }
        if (entry != null) newRows.unshift(entry);
      }
    } else {
      jsonArray.forEach((el) => {
        const elObj = JSON.parse(el);
        if (elObj.event_type == 'alert') {
          newRows.unshift(this.pushAlert(elObj));
        }
      });

      this.setState({ isNew: false });
    }
    if (newRows.length != nrOfAlerts) {
      // Update if theres new entries
      /// Count alerts
      const counts = {};
      newRows.forEach((x) => {
        counts[x[5]] = (counts[x[5]] || 0) + 1;
      });

      Object.keys(counts).forEach((x) => {
        newRowsCount.push([x, counts[x]]);
      });
      // Count alerts
      rows['repeated-table'] = newRowsCount;
      rows['signature-table'] = newRows;

      this.setState({ nrOfAlerts: newRows.length, rows }, () => {
        this.updateSort('repeated-table');
        this.updateSort('repeated-table');
        // BUG needs to run updateSort() twice to avoid elements not being sorted correctly
      });
    }
  });
```

Code listing 6.3 shows how the *cockpit.spawn('tail -f this.logFile')* is monitoring a file implemented in *alerts.jsx*.

Code listing 6.3: *cockpit.spawn([tail -f $logFile])*

```
cockpit.spawn(['tail', '-f', this.logFile], { superuser: 'try' }).stream((content) => {
  if (!content.includes('"event_type":"alert"')) return;
  const { isNew, rows, nrOfAlerts } = this.state;
  const newRows = rows['signature-table'];
  const newRowsCount = [];
  const jsonArray = content.split('\n').filter((x) => x.includes('"event_type":"alert"'));

  if (!isNew) {
    for (let i = 0; i < jsonArray.length; i += 1) {
      let entry;
      try {
        entry = this.pushAlert(JSON.parse(jsonArray[i]));
      } catch {
        console.log(jsonArray[i]);
      }
      if (entry != null) newRows.unshift(entry);
    }
  } else {
    jsonArray.forEach((el) => {
      const elObj = JSON.parse(el);
      if (elObj.event_type == 'alert') {
        newRows.unshift(this.pushAlert(elObj));
      }
    });

    this.setState({ isNew: false });
  }
  if (newRows.length != nrOfAlerts) {
    // Update if theres new entries
    /// Count alerts
    const counts = {};
    newRows.forEach((x) => {
      counts[x[5]] = (counts[x[5]] || 0) + 1;
    });

    Object.keys(counts).forEach((x) => {
      newRowsCount.push([x, counts[x]]);
    });
    // Count alerts
    rows['repeated-table'] = newRowsCount;
    rows['signature-table'] = newRows;

    this.setState({ nrOfAlerts: newRows.length, rows }, () => {
      this.updateSort('signature-table');
      this.updateSort('signature-table');
      // BUG needs to run updateSort() twice to avoid elements not being sorted correctly
    });
  }
});
```

Creating editable forms and tables At the start we tried specifying every configuration setting which was fairly easy with *update.yaml* file being reasonably small, however the *suricata.yaml* file proved to be too extensive and complex to setup. Instead of specifying every form and tables statically, we chose a more dynamic solution to help ease the setup in the "Config" tab.

The solution was to create a recursive function called *setupRecursiveObject* shown in Code listing 6.5. The function traverse through every nested object and builds an array of HTML code that gets returned to be displayed and made editable for the administrator.
The object was built by merging both *suricata.yaml* and *update.yaml* into one big object that our function uses at start. In the function for building the array we perform checks to see if the object path is matching one of our fixed objects called *global.suricataYamlTables* and *global.suricataYamlToggleable* in Code listing 6.4.

If the first object matches the object path the function will construct a table of the array with the right columns specified in *global.suricataYamlTablesColumns*. If the second object matches the object path, it will create an on and off button that administrator can choose to be applied or not in the configuration file.

This function will also make it easier to add new configuration settings in *suricata.yaml* and *update.yaml* followed by newer Suricata releases with only needing to include their new properties in Code listing 6.4.

Code listing 6.4: Defined objects in vars.jsx

```
global.suricataYamlToggleable = {
  pfring: ['bpf-filter', 'bypass', 'checksum-checks'],
  stats: ['decoder-events', 'decoder-events-prefix', 'stream-events'],
};

// Specify which is a table
global.suricataYamlTables = {
  vars: { 'address-groups': [], 'port-groups': [] },
  'rule-files': [],
  'af-packet': [],
  pcap: [],
};
global.suricataYamlTablesColumns = {
  vars: { 'address-groups': ['Name:', 'IP-address:'],
          'port-groups': ['Name:', 'Ports:'] },
  'rule-files': ['Filename'],
  'af-packet': [
    'Interface',
    'Cluster-id',
    'Cluster-type',
    'defrag',
    'use-mmap',
    'mmap-locked',
    'tpacket-v3',
    'ring-size',
    'block-size',
    'block-timeout',
    'use-emergency-flush',
    'checksum-checks',
    'bpf-filter',
    'copy-mode',
    'copy-iface',
  ],
  pcap: [
    'Interface',
    'Buffer-size',
    'bpf-filter',
    'checksum-checks',
    'threads',
    'promisc',
    'snaplen',
  ],
};
```

Code listing 6.5 show how it utilizes some of the declared objects in 6.4 explained earlier.

Code listing 6.5: setupRecursiveObject

```
this.setupRecursiveObject = (obj, objPath, html) => {
  if (html.length == 0) {
    obj.forEach((key) => {
      if (typeof textInputValue[key] == 'undefined') textInputValue[key] = '';
      html.push(
        <FormGroup
          id={key}
          label={key}
          labelIcon={this.setupSwitch(key)}
          style={{
            fontWeight: 'bold',
          }}>
          {typeof _.get(textInputValue, [key]) != 'object' && (
            <TextInput
              id={key}
              value={_.get(textInputValue, [key])}
              onChange={(nil, e) => {
                this.handleTextInputChange(nil, e, [key]);
                this.setState({ updateYamlChanges: true });
              }}
            />
          )}
        </FormGroup>,
      );
      if (typeof _.get(textInputValue, [key]) == 'object')
        this.setupRecursiveObject(_.get(textInputValue, key), [key], html);
    });
    return html;
  }
  const id = objPath.toString().replaceAll(',', '.');
  const indent = objPath.length;

  if (typeof obj == 'object' && obj) {
    if (Array.isArray(_.get(global.suricataYamlTables, objPath))) {
      if (_.get(rows, objPath).length == 0)
        Object.keys(_.get(textInputValue, objPath)).forEach((key) => {
          if (/^[0-9]*$/.test(key)) {
            if (typeof _.get(textInputValue, objPath)[key] == 'object') {
              const buildArr = [];
              _.get(global.suricataYamlTablesColumns, objPath).forEach((key2) => {
                buildArr.push(_.get(textInputValue, [...objPath, key, key2.toLowerCase()]));
              });
              _.get(rows, objPath).push(
                this.setupNewRow(buildArr, _.get(rows, objPath).length, objPath),
              );
            } else {
              _.get(rows, objPath).push(
                this.setupNewRow(
                  _.get(textInputValue, [...objPath, key]),
                  _.get(rows, objPath).length,
                  objPath,
                ),
              );
            }
          } else
            _.get(rows, objPath).push(
              this.setupNewRow(
                [key, _.get(textInputValue, objPath)[key]],
                _.get(rows, objPath).length,
                [objPath],
              ),
            );
        });
      html.push(
        <div
          style={{
            'padding-left': `${indent}em`,
          }}>
          <Table
```

```
            id={id}
            onRowEdit={this.updateEditableRowsnew}
            aria-label="Editable␣Rows␣Table"
            variant={TableVariant.compact}
            cells={_.get(global.suricataYamlTablesColumns, objPath)}
            rows={_.get(rows, objPath)}>
            <TableHeader />
            <TableBody />
          </Table>
          <Button
            icon={<PlusIcon />}
            style={{ marginRight: 'auto' }}
            variant="none"
            className="pf-c-button␣pf-c-plain"
            onClick={() => {
              const emptyBlocks = [];
              _.times(_.get(global.suricataYamlTablesColumns, objPath).length, () =>
                emptyBlocks.push(''),
              );
              _.get(rows, objPath).push(
                this.setupNewRow(emptyBlocks, _.get(rows, objPath).length, objPath),
              );
              this.setState({ rows }, () => {
                this.addTrashToTable(objPath);
              });
            }}
          />
        </div>,
      );
    } else {
      Object.keys(obj).forEach((key) => {
        const name = /^-?\d+$/.test(key) ? obj[key] : key;
        const newPath = [...objPath, key];
        html.push(
          <div
            style={{
              'padding-left': `${indent}em`,
            }}>
            {name.toString()}:
            {typeof _.get(textInputValue, newPath) != 'object' && (
              <TextInput
                id={id}
                value={_.get(textInputValue, newPath)}
                onChange={(nil, e) => {
                  this.handleTextInputChange(nil, e, newPath);
                  this.setState({ updateYamlChanges: true });
                }}
              />
            )}
          </div>,
        );
        if (typeof _.get(textInputValue, newPath) == 'object')
          this.setupRecursiveObject(_.get(textInputValue, newPath), newPath, html);
      });
    }
  }
  return null;
};
```

6.4.1 Remove or add cells to table through user interface

PatternFly version 4 has a feature that can make cells in tables editable, but no built-in functionality options to add or remove cells. To make it possible to add new cells we added a plus icon on every table to give the administrator convenient way to add a new cell. For the remove function we developed a function that is called *addTrashToTable(id)* which adds a trash icon next to the pen in tables shown in Figure 6.13, which can be clicked on to delete the selected cell. This function gets called when:

- The table is created on the page
- The administrator presses the plus icon
- The administrator presses the trash icon

The code works by getting the id of the table displayed on the page as an argument, then uses built-in methods to get the element by id. When found, the table will start iterating through all the cells and insert the icon element next to the pen icon.

Code listing 6.6: addTrashToTable

```
addTrashToTable(id) {
  const { rows, textInputValue } = this.state;
  try {
    _.get(rows, id).forEach((_el, index) => {
      const button = (
        <Button className="pf-c-button␣pf-m-plain" id={`${id}deleteButton${index}`}>
          <TrashIcon />
        </Button>
      );
      const deleteButton = document.createElement('div');
      deleteButton.style.width = '0px';
      deleteButton.onclick = () => {
        // Delete element from table displayed
        _.set(
          rows,
          id,
          _.get(rows, id).filter((_nil, i) => i !== index),
        );
        // Delete element from object list
        const copy = _.get(textInputValue, id);
        const newObject = {};
        Object.keys(copy).forEach((item, loopIndex) => {
          if (index !== loopIndex) {
            newObject[item] = copy[item];
          }
        });
        _.set(textInputValue, id, newObject);
        this.setState(
          () => ({
            rows,
            updateYamlChanges: true,
            textInputValue,
          }),
          () => {
            this.updateTable(id);
          },
        );
      };
      deleteButton.innerHTML = ReactDOMServer.renderToString(button);
      if (
        document.getElementById(`${id}row${index}`) != null &&
        document.getElementById(`${id}deleteButton${index}`) == null
      )
        document.getElementById(`${id}row${index}`).parentElement.appendChild(deleteButton);
    });
  } catch {
    return null;
  }
  return null;
}
```

6.4.2 Spawning processes

In this project we have had the need to spawn processes and retrieve its output. Cockpit makes this easy by providing us with an API to spawn processes [32]. The function that spawns a process is called *spawn()* and it returns a promise that will complete if the process exits successfully, or fail if there's a problem.

One example where we needed to spawn a process is to create a signatures file using touch. Touch is command used in UNIX/linux operating systems to create, change and modify timestamps of a file [45]. In our case we needed to create an empty file. We did not use Cockpit's file API here because it does not provide functionality to create an empty file.

Figure 6.7 shows the code for creating the file with touch. All it does is spawn touch to create a file, and provide feedback to the administrator on success or failure.

Code listing 6.7: createSignatureFile

```
this.createSignatureFile = () => {
  const { textInputValue1 } = this.state;
  cockpit
    .spawn(['touch', `${surRulesPath}/${textInputValue1}`], { superuser: 'try' })
    .done(() => {
      this.addAlert(0, 'success',
                    <span>successfully created file: {textInputValue1}</span>);
    })
    .catch((error) => {
      this.addAlert(
        0,
        'danger',
        <>
          <div>
            <span>Failed creating file: {textInputValue1}</span>
          </div>
          <div>
            <span>Error: {error.message}</span>
          </div>
        </>,
      );
    });
};
```

Chapter 7

Evaluation

In the first part of this chapter we will evaluate our module against all the functional requirements mentioned in the Requirements chapter. In the second part we present the feedback received from the surveys that our client and his colleagues took after testing the module.

7.1 Self evaluation

The following table summarizes whether or not the functional requirements were completed, and detailed descriptions are also presented as below.

Table 7.1: Completion status of functional requirements

Requirement	**Status**
1. The module must allow the administrators to start, stop and restart the Suricata service via the click of the respective buttons on a web interface	Done
2. The module should allow the administrators to view IDS signatures	Partly done
3. The module must enable administration of IDS signatures. The administrators should be able to add signatures by uploading local files or creating new files. The administrators should be able to edit or delete signatures.	Done
4. The module must allow the administrators to download IDS signatures from a vendor and use them.	Done
5. The module must display logs related to the Suricata service.	Done
6. The current status of Suricata service should be displayed.	Done
7. The module must display the alerts generated by the Suricata IDS. The alerts can be sorted by date and time, priority, protocol, category and id.	Done
8. The module must be licensed as open source.	Done

Requirement 1: The module must allow the administrators to start, stop and restart the Suricata service via the click of the respective buttons on a web interface
Our module fulfills this requirement. The Service tab has three buttons (*start*, *stop*, *restart*), which are able to start, stop and restart the Suricata service respectively.

Requirement 2: The module should allow the administrators to view IDS signatures
Our module meets this requirement partly. Administrators can view and edit signatures that are stored locally. The administrator can also view signatures provided by vendors by opening the link next to *vendor* in the "Vendors" subtab which will redirect the administrator to the file containing the signatures. This can be a cumbersome process if the administrator wants to find and view a single signature by itself, and we therefore classify this requirement as partly done.

Requirement 3: The module must enable administration of IDS signatures. The administrators should be able to add signatures by uploading local files or creating new files. The administrators should be able to edit or delete signatures
Our module meets this requirement. Administrators can either create a new file or upload an existing file when adding signatures. These files can be edited or deleted from the interface.

Requirement 4: The module must allow the administrators to download IDS signatures from a vendor and use them
Our module fulfills this requirement. The module allows downloading of IDS signatures from vendors. An URL to the vendor's signature source must be provided.

Requirement 5: The module must display logs related to the Suricata service
Our module meets this requirement. Under the Logs tab, administrators have the option to view systemd logs related to Suricata or the Suricata generated *stats.log* file. The systemd logs can be filtered to display logs from "everything", "current boot", "previous boot", "last 24 hours" or "last 7 days". The tab which displays the *stats.log* content can be set up to compare data from different time frames.

Requirement 6: The current status of Suricata service should be displayed
Our module fulfills this requirement. The current status of Suricata is displayed on the service tab in real-time with colors on the text to help identify the status with a quick glance.

Requirement 7: The module must display the alerts generated by the Suricata IDS. The alerts can be sorted by date and time, priority, protocol, category and id
Our module meets this requirement. The module displays all the alerts generated by Suricata IDS in the Alerts tab and allows them to be sorted by date and time, priority, protocol, category and id. However to prevent the browser from using too much resources, only a set of alerts are displayed at one time. To display older alerts the administrator can click on the "Click to expand" button at the bottom of the table. Depending on the amount network traffic and other factors, thousands of alerts can be generated every second. Hence it is not practical to display all the alerts in a table, instead one should aggregate the data and display it in a useful format. To take a step in that direction our module displays a list of repeated alerts. However our client is currently using an ELK Stack for monitoring alerts triggered by Suricata. With this stack the client already has the ability

to easily search, aggregate and visualise the alerts. For this reason we decided to not prioritize the alerts tab as much as the other tabs. If we had more time we would like to implement much of the ELK stack's functionality into the Suricata module. Additionally we would like to add filtering capabilities so that the administrator can search for specific alerts using the different fields of an alert.

Requirement 8: The module must be licensed as open source
This requirement is met. The module is copyrighted by NTNU SOC and released under the GPL-2.0-or-later [34] license, which is an open source license.

7.2 User evaluation

In this section we discuss the feedback we received from our client and his colleagues. We provided them with a survey and a test environment to test the module. The questions and testing were focused around the functional requirements, and some of the additional features added to the module. Five people tested our module and filled out the survey.

We were most interested to know how easy and understandable the different components of the module were, and as such most of the questions were focused on this. The survey included the possibility to come with more general feedback and specific wishes in a longer textual answer. Table 7.2 describes what the scores for the survey means, while Table 7.3 shows each question we asked in the survey.

Table 7.2: User score description

Score	1	2	3	4	5
Description	Very difficult Very bad Not useful	Difficult Bad Less useful	Neutral Neutral Neutral	Easy Good Somewhat useful	Very easy Very good Useful

Table 7.3: Questions in the survey

1	How well acquainted are you with the administration of Suricata IDS?
2	How easy was it to to start, stop and restart Suricata using our module?
3	How easy was it to find information about the status of Suricata in module?
4	How do you think the presentation of signatures was in our module?
5	How easy was it to add signatures using our module?
6	How easy was it to edit signatures using our module?
7	How easy was it to delete signatures using our module?
8	How easy was it to fetch signatures from a vendor?
9	What do you think of the way we present logs?
10	What do you think of the way we present alarms?
11	How easy was it to configure Suricata using our module?
12	How useful did you find the tooltips given by the module?

13	How was your experience using the module?
14	Which web browser did you use when testing the module?
15	What is your overall impression of the module? Feel free to come up with suggestions for what could have been better. Are there any other features you would like to see in the module?

The first thing we wanted to know was each of the participants experience with the administration of Suricata IDS (i.e., Question 1 listed in Table 7.3). This was done in order to gain an understanding of whether administrators having less experience with Suricata could use the module effectively. Most of the participants answered that they were well acquainted with administration of Suricata.

All of the participants reported that it was very easy to start, stop, and restart the Suricata service from our module (i.e., Question 2). Similarly, we got good feedback on whether or not it was easy to find information about the status of the Suricata service (i.e., Question 3). Two of the participants answered that they experienced it as easy while three answered that it was very easy.

In question 4, we wanted to know whether the presentation of signatures in the module was satisfactory. One of the participants reported that they were neutral, while three of them thought it was good, and one very good. The add, edit and delete signature features (see Questions 5-7) were mostly well received and thought of as easy to use with an average score of 4 out of 5. All of the participants reported that it was very easy to fetch signatures from a vendor (i.e., Question 8), which was no surprise considering all the administrator has to do is press the ”Fetch vendors” button.

In question 9, We asked the participants what they thought about the way we present logs. Here the average score was 4,4 out of 5 and falls into the category between good and very good. On the similar question regarding alarms (i.e., Question 10), the participants generally thought the way we present alarms is good.

On a question on how easy it was to configure Suricata using our module (i.e., Question 11), the average score was 4.2 out of 5 meaning that it was easy to very easy for the participants. The standard deviation range for Question 12 is considerably big, which tells us that most participants found tooltips to be somewhat useful and some others less useful. The average score was 3,8 meaning it was somewhat useful.

Figure 7.1 shows the average score and standard deviation for the short answer questions from the survey.

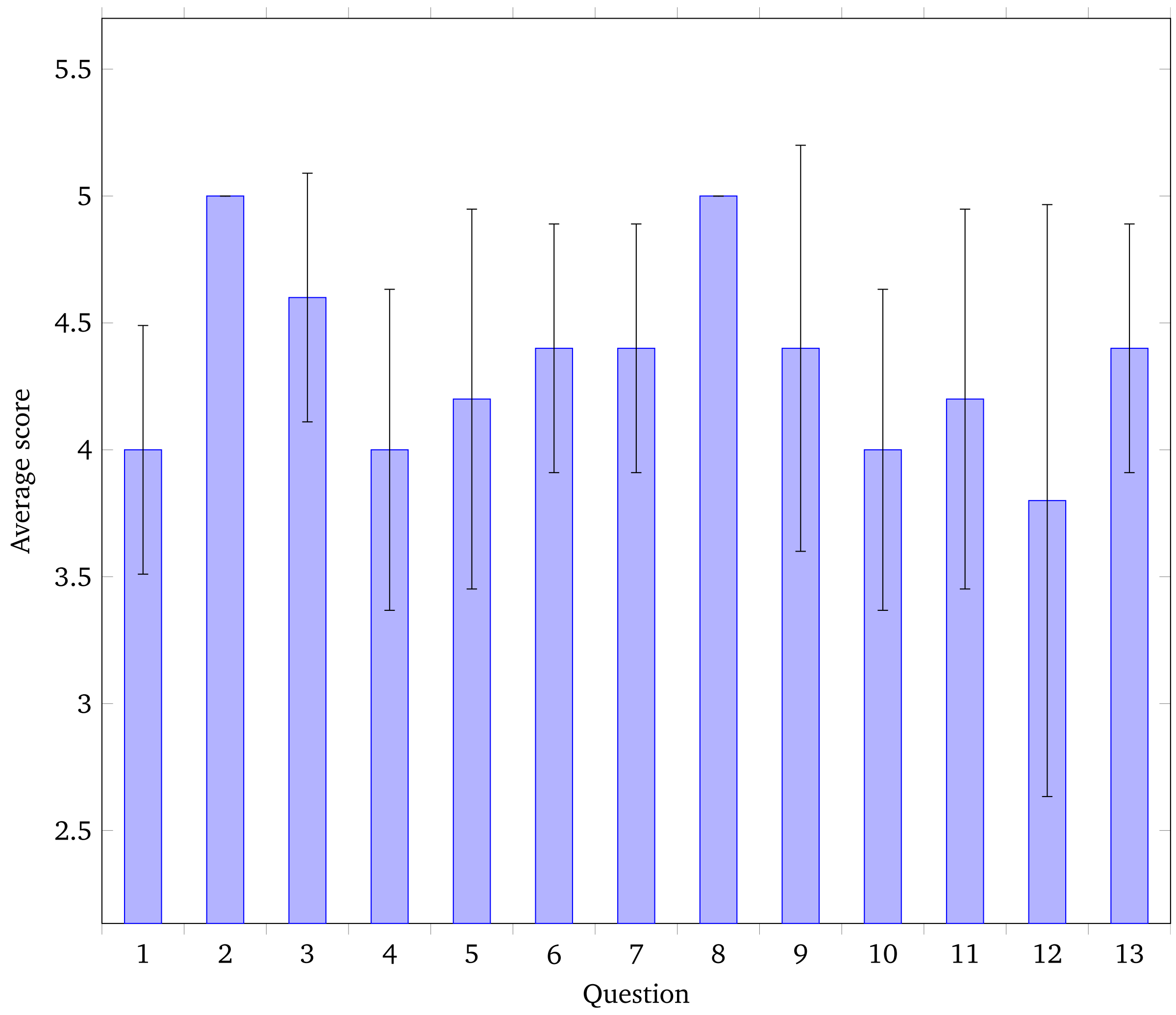


Figure 7.1: Survey score graph

In Question 13 we wanted to know the overall experience the participants had when using the module. The average score turned out to 4,4 out of 5 which falls into the good to very good category.

We asked the participants which web browser they used when testing the module (i.e., Question 14). This was to gain a certain idea of the module's performance on different browsers. The participants used Safari, Google Chrome, Microsoft Edge and Mozilla Firefox.

In the last question we wanted the participants to give a more comprehensive answer about their overall impression of the module, and whether they had any suggestions for improvements or additional features that could be added. The participants answered as follows:

- **Respondent 1**:
  Overall good, some features a little unfinished.

- **Respondent 2**:
  This seems like a nice module. It seems to simplify administration of a single sensor. It would be nice to store the configuration in a git-repo as well, so it can be replicated across multiple sensors.

- **Respondent 3**:
  No answer

- **Respondent 4**:
  I love the idea of doing the configuration and ID management from a graphical interface. I think it has great potential to improve the general understanding of how Suricata is configured and present alerts in such a way that is more understandable by the mass and not just by people with deep understanding of the service. I don't think that being able to configure Suricata in the interface as it is now is necessarily better or worse. You still need to understand how Suricata work and what it is you wish to achieve. Also if you administrate Suricata, chances are you also know how to use a terminal and flat text configuration files. So unless there is something added like for example an easy to reach description of what each setting does and how it can be used or validation/error checking individual settings, the GUI does not add anything in terms of how easy it is to configure of Suricata. I still love the idea and it has massive potential, but I personally feel the potential lays in automating adding/removing rules, adding rules manually and helping the person "choose the right things" when creating a rule and fleet management (being able to see data/rules for all the Suricata nodes in your network).

- **Respondent 5**:
  I would like an easy way to disable a rule. A way to see each rule with in the rulefiles could be nice, maybe with a view of alerts from said rule.

The general response was mostly positive and indicated to us that we were on the right track. The idea of doing the configuration and ID-management of Suricata from a graphic interface was well received and believed to have great potential in improving the general understanding of how Suricata is configured.

From the feedback we gathered that the current version of the module does not necessarily make the configuration of Suricata better or worse. It is still necessary for the administrator to understand how Suricata works, and know what they want to achieve. Another good point was that unless better descriptions of what each setting does and how the setting can be used, the graphical user interface does not make the configuration of Suricata any easier. This is especially true for administrators who already have experience with Suricata as they are already familiar with using a terminal and flat text configuration files.

One participant told us that some features seemed a little unfinished which is a fair assessment as some features in the Config and Alerts tab could need a fair bit of work. One suggestion we received was the ability to store the configuration in a git repository so that it could be replicated across multiple sensors. Another participant suggested an easier way to disable a rule, and the possibility to see each rule within the signature files, perhaps with a view of alerts from said rule.

Chapter 8

Closing Remarks

In this chapter we will discuss our thoughts on the project as a whole and talk about what we have learned.

8.1 Discussion and learning outcome

The group members have worked as a group in earlier projects and as such knew each others strengths and weaknesses, and how we work as a group. There has always been good communication between the members which resulted in a constructive work environment, with no major disagreements within the group. The disagreements that did occur were focused on specific project related choices and were handled by voting.

Due to the pandemic the group members are located in different cities. The group made a decision to work remotely using different tools like git, overleaf and discord, described in Chapter 4. However we agreed to meet physically if working remotely caused any major issues. Fortunately for us there were no such issues.

The group had regular meetings with the supervisor every Wednesday, where we presented our weekly progress. These meetings were extremely important as the group received feedback on possible improvements. To ensure that we were developing a module that met the client's expectations, we had occasional meetings with our client. Additionally the group met three times per week to work collectively. These meetings were valuable because the members could share their thoughts on how to solve the different tasks, and make decisions together as a group. Since we worked in parallel on the development and thesis writing, the meetings served to update each other on the progress made to either task.

To distribute the tasks we initially split up the group where two members focused on the development while the other two focused on writing the thesis. However there were times the group needed more focus on development, and times where more focus was needed on writing the thesis. In these situations adjustments were made to have three or all members work on development or writing the thesis. In our weekly meetings we discussed if we needed to add or reprioritize tasks.

One of the biggest challenges our group had was time management as we had initially miscalculated how much time we would need to develop the module. This meant that we had to delay the writing of some parts of the thesis until the necessary functionality in the module was ready. This was especially true for the implementation chapter as many of the pictures and code snippets would be outdated by the time they were written.

The main reasons for the delay on the development was due to scope creep and the fact that no one on the group had any experience with React prior to this project. While developing the module we saw that we could add features that would improve the usability of the module. This led to us spending a lot of time developing these features. At some point we decided on a new deadline for the development and made a priority list of what was being worked on.

In our project plan we had originally planned to take a testing first approach to the development. However, we abandoned this approach as we needed to allocate more time to develop the module.

None of our members had much experience with the technologies we had to use and as such had to spend some time early in the project to brush up on those. We knew about some of the fundamentals behind Intrusion Detection Systems, but had never actually used one before this project. Diving into the intricacies of Suricata helped us gain a better understanding of IDS. React is a framework none of the group members had used before, and as such had to spend some time to learn about and get accustomed to.

8.2 Conclusion

The main goal of the project was to develop a module for the web-based interface Cockpit that would simplify the management of Suricata IDS. After nearly six months of work on the bachelor project we can confidently say that we are happy with the resulting product. The module meets all the initial requirements as well as some additional features that we believe increases the general usefulness of the module.

There is no doubt that the module can be improved upon and we have listed our suggestions for improvements in the next section ”Future Work”. All the new technologies we learned and experience we gained by working on this project will undoubtedly serve us well in the future.

8.3 Future Work

The module is finished and functional as per the requirements set by our client, however there are possibilities for some improvements and additions that could make the module even better. These improvements came to us during development and in conversations with the client, but had to be set aside due to time constraints.

- **Add error checking when process output stream prints standard error**
  When the administrator presses the ”apply changes” button, the spawned process will be out-

put in a window for the administrator to see how the process is doing. This can be achieved by redirecting standard error to its own stream output notifying the administrator.

- **Improve Config tab code**
  The Config tab is currently acting as a prototype since it was not one of our initial requirements. The code handling reading ”suricata.yaml” and ”update.yaml” recursively is not written optimally, so a rewrite could help readability and clear out some bugs.

- **Add outline and a search bar in config tab**
  Covering all the ”suricata.yaml” config options fill the page with too many options making it unorganized and overwhelming to use. An idea to fix this is to add an outline bar to help navigate the administrator and add a search bar for the more experienced administrator to quickly filter out the unwanted options.

- **Integrate Puppeteer with Jest**
  For now the tests needs to be executed manually. By integrating our tests with Jest it would help managing our tests better without doing it manually.

- **Add more testing**
  Currently we have only written one test that checks for three things:
  - Logging into Cockpit
  - Select Suricata module
  - Tests all buttons in Service tab

  We planned to have tests on every tab and their subtabs, but due to time constraint we only managed to create one test that can be used as a template to help develop more tests.

- **”Create file” adds .rules extension on its own**
  In the current implementation the administrator has to add the ”.rules” extension manually when creating a new file. As there are no other extensions allowed in this specific case we could save the administrator some time by adding the extension automatically.

- **Visual representation**
  We really wanted to represent some of the data in a more meaningful way. This would be the case for statistical records from Suricata’s ”stats.log” file. Data like packet loss, kernel loss, memory usage and alerts detected, etc., could be shown in graphs so that it is easier to get a quick overview of what is happening.

- **Tool tips**
  Another nice addition would be to have the possibility to hover over e.g., the counters in the statistics to get a short summary of what they mean. This would be helpful for administrators not as familiar with Suricata.

- **Better presentation of alerts**

For now the alert tab is almost featureless when it comes to the visual presentation of alerts and filtering. It would be tremendously helpful for the administrator to have a way to filter displayed alerts by having the ability to search for alerts and specify matching columns such as date, priority, ports, and the source or destination of the IP-address. To represent the data in a simplified manner, we wanted to use diagrams that are interactive with the filter bar to display alerts on a diagram. The diagram could also be exportable as an image file.

- **Exportable data**
  To assist the administrators with statistics, it would be helpful to make alerts exportable to a CSV file. These can then be processed on other math software. The filter ability can also help with separating unwanted data from the CSV.

- **Localize signatures**
  With many signatures active it can be a struggle to manage and fix potential signature issues. By making every cell in the "Id" column interactive such that clicking them would point the administrator to where the signature originates from, either a local rule file or a vendor, it could help administrators localize the signature. This feature would allow the administrator to easily find where the signature is defined and take further actions (e.g., edit the signature).

Bibliography

[1] E. B.V., *What is the elk stack?* https://www.elastic.co/what-is/elk-stack, n.d.

[2] *What is gnu/linux?* https://www.techopedia.com/definition/15759/gnulinux, n.d.

[3] *Introducing json*, https://www.json.org/json-en.html, n.d.

[4] Linode, *What is systemd?* https://www.linode.com/docs/guides/what-is-systemd/, 2020.

[5] Wikipedia contributors, *Url — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=URL&oldid=1015459310, [Online; accessed 20-May-2021], 2021.

[6] pQd, *What is the difference between unix sockets and tcp/ip sockets?* https://serverfault.com/a/124518, 2010.

[7] suricata, *Suricata*, https://suricata-ids.org/, n.d.

[8] cockpit-project, *Cockpit*, https://cockpit-project.org/, n.d.

[9] Distrowatch, *Major distributions*, https://distrowatch.com/dwres.php?resource=major, n.d.

[10] W. contributors, *Intrusion detection system*, https://en.wikipedia.org/w/index.php?title=Intrusion_detection_system&oldid=1015627231, 2021.

[11] W. contributors, *Host-based intrusion detection system*, https://en.wikipedia.org/w/index.php?title=Host-based_intrusion_detection_system&oldid=1015627294, 2021.

[12] M. Rezek, *What is the difference between signature-based and behavior-based intrusion detection systems?* https://accedian.com/blog/what-is-the-difference-between-signature-based-and-behavior-based-ids/, 2020.

[13] Bricata, *What is suricata? intro to a best of breed open source ids and ips*, https://bricata.com/blog/what-is-suricata-ids/, 2021.

[14] O. I. S. Foundation, *Suricata rules*, https://suricata.readthedocs.io/en/suricata-6.0.2/rules/intro.html, n.d.

[15] O. I. S. Foundation, *Suricata rules*, https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Rules, n.d.

[16] O. I. S. Foundation, *Suricata yaml*, https://suricata.readthedocs.io/en/suricata-6.0.2/configuration/suricata-yaml.html?highlight=suricata.yaml#action-order, n.d.

[17] O. I. S. Foundation, *6.1. rules format*, https://suricata.readthedocs.io/en/suricata-6.0.0/rules/intro.html#rule-options, n.d.

[18] O. I. S. Foundation, *6.2.1. msg (message)*, `https://suricata.readthedocs.io/en/latest/rules/meta.html#msg-message`, n.d.

[19] O. I. S. Foundation, *10.1. suricata.yaml*, `https://suricata.readthedocs.io/en/latest/configuration/suricata-yaml.html`, n.d.

[20] O. I. S. Foundation, *Eve json output*, `https://suricata.readthedocs.io/en/suricata-6.0.0/output/eve/eve-json-output.html`, n.d.

[21] O. I. S. Foundation, *Statistics*, `https://suricata.readthedocs.io/en/suricata-6.0.0/performance/statistics.html`, n.d.

[22] O. I. S. Foundation, *10.1.8.10. stats*, `https://suricata.readthedocs.io/en/suricata-6.0.0/configuration/suricata-yaml.html?highlight=stats.log#id1`, n.d.

[23] O. I. S. Foundation, *7.1.3. controlling which rules are used*, `https://suricata.readthedocs.io/en/suricata-6.0.0/rule-management/suricata-update.html?highlight=suricata.rules#controlling-which-rules-are-used`, n.d.

[24] O. I. S. Foundation, *Suricata-update - update*, `https://suricata-update.readthedocs.io/en/latest/update.html`, n.d.

[25] O. I. S. Foundation, *Suricata-update*, `https://github.com/OISF/suricata-update#files-and-directories`, n.d.

[26] O. I. S. Foundation, *Rule matching*, `https://suricata-update.readthedocs.io/en/latest/update.html#rule-matching`, n.d.

[27] O. I. S. Foundation, *Example configuration files*, `https://suricata-update.readthedocs.io/en/latest/update.html#example-configuration-files`, n.d.

[28] cockpit-project, *Running cockpit*, `https://cockpit-project.org/running.html`, n.d.

[29] OISF, *Index.yaml*, `https://www.openinfosecfoundation.org/rules/index.yaml`, n.d.

[30] D. Radigan, *What is kanban?* `https://www.atlassian.com/agile/kanban`, n.d.

[31] Git, *Git*, `https://git-scm.com/`, n.d.

[32] cockpit-project, *Cockpit.js: Spawning processes*, `https://cockpit-project.org/guide/latest/cockpit-spawn`, n.d.

[33] cockpit-project, *Api: Base1*, `https://cockpit-project.org/guide/236/api-base1.html`, n.d.

[34] W. contributors, *Gnu general public license*, `https://en.wikipedia.org/w/index.php?title=GNU_General_Public_License&oldid=1015788252`, 2021.

[35] B. Smith, *A quick guide to gplv3*, `https://www.gnu.org/licenses/quick-guide-gplv3.html`, 2007.

[36] Wikipedia contributors, *Mit license — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=MIT_License&oldid=1014888256`, [Online; accessed 20-May-2021], 2021.

[37] M. Pitt, *Cockpit-project / starter-kit*, `https://github.com/cockpit-project/starter-kit`, n.d.

[38] webpack.js.org, `https://webpack.js.org/`, n.d.

[39] GNU.org, *Gnu make*, `https://www.gnu.org/software/make/`, 2020.

[40] O. Foundation and other contributors, *Find and fix problems in your javascript code*, `https://eslint.org/`, n.d.

[41] Wikipedia contributors, *React (javascript library) — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=1017634608`, [Online; accessed 20-May-2021], 2021.

[42] reactjs, *About patternfly*, `https://www.patternfly.org/v4/get-started/about`, n.d.

[43] Puppeteer, *Puppeteer*, `https://pptr.dev/`, n.d.

[44] cockpit-project, *Contributing*, `https://cockpit-project.org/external/wiki/Contributing.html`, n.d.

[45] B. Rani, *Touch command in linux with examples*, `https://www.geeksforgeeks.org/touch-command-in-linux-with-examples/`, 2021.

Appendix A

Additional Material

A.1 Project Survey

Cockpit module for administration of Suricata IDS

We would like your feedback regarding the module.
The survey should not take more than 10 minutes :)

Answers with a scale from 1 to 5 have the following representation:

1	2	3	4	5
Very difficult	Difficult	Neutral	Easy	Very easy
Very bad	Bad	Neutral	Good	Very good
Not useful	Less useful	Neutral	Somewhat useful	Useful

* Required

1. How well acquainted are you with the administration of Suricata IDS? *

1	2	3	4	5
○	○	○	○	○

2. How easy was it to to start, stop and restart Suricata using our module? *

1	2	3	4	5
○	○	○	○	○

3. How easy was it to find information about the status of Suricata in module? *

1	2	3	4	5
○	○	○	○	○

4. How do you think the presentation of signatures was in our module? *

1	2	3	4	5
◯	◯	◯	◯	◯

5. How easy was it to add signatures using our module? *

1	2	3	4	5
◯	◯	◯	◯	◯

6. How easy was it to edit signatures using our module? *

1	2	3	4	5
◯	◯	◯	◯	◯

7. How easy was it to delete signatures using our module? *

1	2	3	4	5
◯	◯	◯	◯	◯

8. How easy was it to fetch signatures from a vendor? *

1	2	3	4	5
◯	◯	◯	◯	◯

9. What do you think of the way we present logs? *

1	2	3	4	5
◯	◯	◯	◯	◯

10. What do you think of the way we present alarms? *

1	2	3	4	5
○	○	○	○	○

11. How easy was it to configure Suricata using our module? *

1	2	3	4	5
○	○	○	○	○

12. How useful did you find the tooltips given by the module? *

1	2	3	4	5
○	○	○	○	○

13. How was your experience using the module? *

1	2	3	4	5
○	○	○	○	○

14. Which web browser did you use when testing the module?

Enter your answer

15. What is your overall impression of the module? Feel free to come up with suggestions for what could have been better. Are there any other features you would like to see in the module?

Enter your answer

Submit

Never give out your password. Report abuse

Table A.1: User testing result from 5 participants

Survey participants	**#1**	**#2**	**#3**	**#4**	**#5**
How well acquainted are you with the administration of Suricata IDS?	5	4	3	3	5
How easy was it to start, stop and restart Suricata using our module?	5	5	5	5	5
How easy was it to find information about the status of Suricata in module?	4	5	5	4	5
How do you think the presentation of signatures was in our module?	4	4	5	3	4
How easy was it to add signatures using our module?	5	4	5	3	4
How easy was it to edit signatures using our module?	4	4	5	5	4
How easy was it to delete signatures using our module?	4	4	5	5	4
How easy was it to fetch signatures from a vendor?	5	5	5	5	5

What do you think of the way we present logs?	3	5	5	4	5
What do you think of the way we present alarms?	3	5	4	4	4
How easy was it to configure Suricata using our module?	4	5	4	3	5
How useful did you find the tooltips given by the module?	5	4	3	2	5
How was your experience using the module?	4	5	4	4	5
Which web browser did you use when testing the module?	Safari, Chrome, Microsoft Edge	Firefox		Safari	Firefox

What is your overall impression of the module? Feel free to come up with suggestions for what could have been better. Are there any other features you would like to see in the module?	Overall good, some features a little unfinished.	This seems like a nice module. It seems to simplify administration of a single sensor. It would be nice to store the configuration in a git-repo as well, so it can be replicated across multiple sensors.		I love the idea of doing the configuration and ID management from a graphical interface. I think it has great potential to improve the general understanding of how Suricata is configured and present alerts in such a way that is more understandable by the mass and not just by people with deep understanding of the service.I don't think that being able to configure suricata in the interface as it is now is necessarily better or worse. You still need to understand how Suricata work and what it is you wish to achieve. Also if you administrate Suricata, chances are you also know how to use a terminal and flat text configuration files. So unless there is something added like for example an easy to reach descriptiopn of what each setting does and how it can be used or validation/error checking individual settings, the gui does not add anything in terms of how easy it is to configure of Suricata.I still love the idea and it has massive potential, but I personally feel the potential lays in automating adding/removing rules, adding rules manually and helping the person "choose the right things"" when creating a rule and fleet management (being able to see data/rules for all the Suricata nodes in your network)."	I would like an easy way to disable a rule. A way to see each rule with in the rulefiles could be nice, maybe with a view of alerts from said rule.

A.2 Project Proposal

Cockpit modul for administrasjon av Suricata IDS

Seksjon for Digital sikkerhet ved NTNU har ansvaret for deteksjon, sikkerhetsanalyse og hendelseshåndteringen ved NTNU. Over en periode siden 2016 har seksjonen bygget opp et sensornettverk for å oppdage brudd på sikkerheten til NTNU hvor en av hovedkomponentene i denne løsningen er basert på det åpne kildekode intregningsdeteksjonssyetemet Suricata [1]. Seksjonen er en partner i et samarbeidsprosjekt for å etablere et nasjonalt cybersikkerhetssenter for kunnskapssektoren døpt «Cybersikkerhetssenteret for Forsking og utdanning» eller kortversjonen «EduCSC-NO». Dette prosjektet er et samarbeid imellom Uninett AS, UiO CERT og NTNU SOC. Modulen ønskes utviklet med bakgrunn av arbeidspakken «Sensorplattform» som har som mål å bygge den neste generasjonen av nettverksbaserte IDS-sensorer for kunnskapssektoren.

Oppdragsgiver

Seksjon for Digital sikkerhet ved NTNU har ansvaret for deteksjon, sikkerhetsanalyse og hendelseshåndteringen ved NTNU. Oppdraget gis på vegne av satsningen ved å etablere «Cybersikkerhetssenteret for Forskning og utdanning».

Kontaktperson 1 (Hovedkontaktperson):
Navn: Christoffer Vargtass Hallstensen
Tittel: Gruppeleder, NTNU SOC
Epost: christoffer.hallstensen@ntnu.no
Tel: 611 35 145 / 481 35 180

Kontaktperson 2:
Navn: Arne Øslebø
Tittel: Senior teknisk arkitekt, Uninett AS
Epost: arne.oslebo@uninett.no

Oppgavens mål

Oppgaven går ut på å utvikle en plugin/modul til administrasjonsgrensesnittet cockpit (https://cockpit-project.org) som forenkler administrasjon av Suricata IDS/NSM (https://suricata-ids.org) sensorer ved å tilby et grafisk brukergrensesnitt som kan aksesseres i en nettleser. Målet med dette er å senke terskelen for at sluttbruker kan gjøre enkle administrative oppgaver på sensorer plassert ute i egen organisasjon.

Oppgavens krav

Modulen som utvikles må tilfredsstille følgende krav:

- Starte, restarte og stoppe tjenesten suricata
- Kunne administrere IDS-signaturer
- Vise helsetilstanden til tjenesten
  Vise relevante tjenestelogger
- Vise alarmer
- Produktet må lisensieres som åpen kildekoden

A.3 Project Plan

Project Plan

Sindre Morvik, Anders Svarverud,
Sigve Sudland, Said-Emin Letsjievitsj Evmurzajev

February 2, 2021

Contents

1 Background and goals

1.1 About us

We are four students in our last semester of the study program IT-Operations and Information Security at the Norwegian University of Science and Technology (NTNU) in Gjøvik. We are a varied group with many different interests and therefore, have some differences in which subjects we chose. We see the varied experiences of our members as a strength in a big project like this.

1.2 Background

Section for Digital Security at NTNU (NTNU SOC) has since 2016 built a sensor network based on the open source Intrusion detection system (IDS) Suricata to help with identifying security incidents. NTNU SOC is currently in cooperation together with Uninett AS and UiO CERT on a project to establish a national cyber security center for the education sector named "Cybersikkerhetssenteret for Forskning og Utdanning". One part of this project is the "Sensor platform" which has as a goal to build the next generation of network based IDS-sensors for the education sector. The Cockpit module for Suricata IDS is developed as a part of this.

Cockpit is an interactive server administration interface. It is free, easy-to-use, integrated, glanceable, and open web-based.[1]
It is mainly used to provide a graphical web interface for services that are otherwise managed from a terminal.

An Intrusion Detection System (IDS) is described by Wikipedia as "a device or software application that monitors a network or systems for malicious activity or policy violations." [2] When suspicious activity is discovered by the system an alert is generated and sent to the administrators.

1.3 Project Goals

This project should result in a Cockpit module for administrating Suricata, and is developed with the goal of making the administration easier by implementing a graphical web interface. The project should also result in a thesis that reflects on the process of creating the module, and the decisions that were made.

2 Scope

2.1 Subject Area

The project will touch several different areas within software engineering, web-development, security and IT-Operations fields.

- Developing a Cockpit module
  - Create a plugin for the Cockpit User Interface to easily manage Suricata.
- DevOps CI/CD
  - Develop pipeline for testing newly committed code pushed by a developer by running real life scenario testing procedures and validating the output before deployment.
- Web design and development
  - Cockpit plugin uses Javascript, HTML and CSS files for structure and layout.
- Information security
  - Avoid creating vulnerabilities while developing the module.

2.2 Task Description

Our task is to develop a module for the web-based interface tool for server administration, Cockpit, that will simplify the administration of Suricata IDS. The end product should be a graphical user interface for Suricata that can be accessed in a web browser. The goal of the project is to reduce the threshold for end users to do simple administrative tasks on sensors within their own organizations. The end user can vary from organization to organization, but in this project we will assume that the end user is a system administrator.

2.3 Requirements

2.3.1 Functional requirements

1. The module should allow the user to start, stop and restart the Suricata service via the click of the respective buttons.
2. The module should allow the user to view IDS signatures in a readable format.
3. The module should enable easy administration of IDS-signatures. IDS-signatures are the rule sets/patterns used by the IDS to detect suspicious activity.
4. The module should allow the user to easily download IDS signatures from a vendor and use them.
5. The module should display logs related to the Suricata service.
6. It should be possible to search and sort the log entries using the fields a log entry consists of (e.g. priority, time).
7. The status of Suricata service should be displayed on the cockpit web interface.
8. The module should display the alarms generated by the IDS.
9. The module should allow the displayed alarm entries to be searched and sorted, using fields the entry consists of.

2.3.2 Non-functional requirements

2.3.2.1 Compatibility

Cockpit is supported and tested on the following operating systems: [3]

- Fedora
- Red Hat
- Fedora CoreOS
- Project Atomic
- CentOS
- Debian
- Ubuntu

Cockpit is developed for and routinely tested for the following web browsers:[3]

- Firefox
- Google Chrome
- Microsoft Edge

With this in mind we will take the same approach and focus our development for those three browsers. According to the documentation[3] Cockpit *might* also work with other browsers.

2.3.2.2 Reliability and maintainability

In order to increase the maintainability of the module, we will strive to have good documentation and comments in the code. Taking a modular approach to the code will also improve maintainability.

By following best practices and taking a testing first approach we will reduce the chance of us making mistakes and introducing bugs in the code.

2.3.2.3 Security

By following coding and security best practices we will reduce the attack surface of the module.

2.3.2.4 Usability

The underlying goal of the module is to make it easy to use. Our thought process is that end users should be able to administrate Suricata with minimal knowledge of the service. To make this possible we will leverage research done on the topic of web design. Cockpit's interface is already a good starting point and we will attempt to design our module in a way that blends well with that. An intuitive graphical interface will go a long way to ease up the demanding technological knowledge otherwise needed to administrate Suricata through the terminal. We will implement input validation to ensure that valid values are entered, and provide suitable error messages.

2.4 Limitations

2.4.1 Operating System

To streamline the development of the module, we have decided to use Ubuntu as the main development platform. We will not take other Linux distributions into consideration, but this should not have any serious impact on the end product. We will use the system and service manager systemd [4], for information regarding the status of services. This might cause some troubles on more obscure Linux distributions that do not support systemd, but it should not be a problem on most popular distributions.

2.4.2 Testing in production environment

As we do not have access to the live sensors placed in NTNU's locations we can only test the module using publicly available pcap files to simulate network traffic. This should work fine for our purposes, despite not having the possibility to test the module in the production environment.

2.4.3 BitBucket pipeline

In our student BitBucket plan subscription we are currently limited to 500 build minutes per month. Instead of running the pipeline for every commit, we have changed it to run on pull requests to conserve the time.

3 Project Organization

3.1 Roles and Responsibilities

Project Leader: Sindre Morvik
Vara Leader: Sigve Sudland
Secretaries: Said-Emin Evmurzajev and Anders Fjeldheim Svarverud

3.2 Workflow and Group Rules

3.2.1 Workflow

We use Clockify to keep track of time spent by each member on different tasks related to the project. All tasks that need to be done will be added to Trello and each member moves the task they are working on to the "In Progress" stage. After a member is done working with a task they will move it to the "Peer Review" stage where every member will quality check each others work. After the "Peer Review" is done, the task will be moved on to "In Test" stage where we try to integrate it into our test environment. When everything is working like it should the task is finally moved to the "Done" stage.

When a developer wants to push changes to our git repository in *Bitbucket* they will need to work on a separate branch and create a pull request for other members to review. During the review of the pull request a CI pipeline will be automatically started to validate the new changes does compile without warnings/errors, linting and acceptance testing. It is very important for the developer to create meaningful commit messages that explains the changes like fixes, typos and added/removed features. This way we can quickly track down the broken changes.

We have created a Discord channel that we use for voice and text communication between the group member. Communication with our client and our supervisor is to be done via Microsoft Teams, meetings included.

3.2.2 Group Rules

3.2.2.1 Scheduled Meetings Every Monday from 12:30 the group meets for a status report and to make ready for the meeting with our supervisor. Each Wednesday we meet with our supervisor 11:00 - 11:30. After the meeting with our supervisor we will again meet to discuss the feedback we got from 11:30-12:30. Each Friday we meet to work together from 10:00-17:00. Every member can call in for extra group meetings should they see the need for it. There should be a minimum of 2 days notice for meetings outside the regular scheduled plan.

3.2.2.2 Workload Every group member is expected to work between 25 - 30 hours per week until the project deadline.

3.2.2.3 Absence If a member is not able to show up to meetings or is unable to work due to e.g. illness the group should be notified in advance. If necessary tasks will be assigned to another group member.

3.2.2.4 Securing the project report We will use Overleaf's integrated version control system to keep track of changes to the project report and related documents. The group will also sync the overleaf project to a GitHub repository which will act as a secondary backup.

3.2.2.5 Logging Every group member logs their time used on the project on Clockify, a simple time tracker and timesheet app. We will keep meeting minutes from our weekly meetings with our supervisor, from every meeting we have with our client, and from our own scheduled meetings. Additionally everyone in the group has to write a recap of all the work they do during the week.

3.2.2.6 Rule violations If a member commits a violation of the rules, he will be notified by the other members of the transgression. Repeated violations will result in a written and oral warning. If any member receives three warnings, all members will be called in for a meeting with our supervisor. In the meeting we will discuss what needs to be done to prevent further violations. All warnings given to members of the group must be documented.

4 Planning and Reporting

4.1 Main Project Sections

4.1.1 Development Model

When deciding on a development model we discussed using either Waterfall, Scrum, eXtreme Programming or Kanban. Due to the nature of our project we decided to use Kanban with some elements from eXtreme Programming. The reasoning behind our choice was that it is very agile allowing new tasks to be added to the backlog and to be worked on immediately. In contrast to scrum that has us locked into a cycle which we did not find beneficial for this project. We also decided multiple functions can be developed in parallel, which makes the sequential waterfall model unsuitable for us.

Kanban has a concept called WIP limit. If a state/column e.g. "in progress" has a WIP limit of three that column can not have more than three cards (i.e. tasks) in it. When a column is maxed out the team focuses on those cards so they can be moved forward. The WIP limits expose bottlenecks in the workflow and can be used to improve the workflow and make it more efficient [5]. At the moment we have decided to put WIP limit of eight on the work in progress stage, WIP limit of four on the peer review stage and a WIP limit of three on the testing stage. It is the group's first time using Kanban, and it is hard to decide the limits that will work best for us, therefore, these limits might be modified later on to better fit us.

We also chose to include test-driven development from eXtreme programming in our Kanban development model [6]. Test driven development focuses on writing the tests based on the client's requirements before writing the code.

4.1.2 Method and Approach

The final module should provide a graphical web interface. To achieve this, we will first make a mock-up of how we want the interface to look. The mock-up will then be shown to our clients for them to provide us with some feedback as to how they think it should be. From the feedback we can ascertain a shared vision of how the final product will look. The work on the interface will most likely be a very iterative process with several additional changes along the way. We will follow this same process until we have an interface that will satisfy the client and provide end users with an easy entry-point into Suricata administration.

As for the development and testing we will have to be careful. Due to confidentiality concerns we might not be able to gain access to the real data gathered from NTNU's use of Suricata. Because of this we will use our own networks or downloaded pcap files to simulate an environment in which Suricata runs, at least in the start of the development. As we go further along, we could have a need for data from NTNU SOC in order to properly satisfy the requirements. In this case we will communicate our wishes to the client and see if something can be arranged.

4.2 Status Meetings and Decision Points

Every Wednesday we have a short meeting with our supervisor Jia-Chun Lin (hereby Kelly) from 11:00 - 11:30. In these meetings we will present what we have done during the week, and what we plan to do going into the next meeting. Kelly will offer us advice, steer us in the right direction and answer questions should we have any. Every Monday from 12:30 to 13:30 the group will meet and discuss what each of us did in the previous week. After each guidance meeting the group will meet again to discuss the plan going forward and any decision points will be made at this meeting. We have a direct communication line to our client that can be used to ask questions that might not need a face-to-face meeting. The group can call the client to meetings on Teams if necessary.

5 Quality Control

5.1 Documentation, Standards and Source Code

All work relevant to the bachelor thesis should be documented in the project report. Each member must also comment their code to make it more readable and easier to use.

The group will follow best practices described for each of the technologies we use. This includes standards for commenting and structuring the code. For Javascript we will follow the JSDoc commenting convention [7]. For the HTML and CSS we will follow W3C standards. We will use LaTeX templates for Bachelor Thesis provided to us by NTNU[8].
In addition we will use Cockpit's starter kit[9] to make sure we follow the standard/recommended steps for building a module. It allows us to focus on development rather than focusing on all the small details.

5.2 Configuration Management

We will use Overleaf which is a version control system to write our thesis. It allows us simultaneously write on the thesis as well as keep track of the changes. For code management Git with Bitbucket will be used, which allows us to collaborate and keep track of changes in an efficient manner. We will also be using the pipeline functionality Bitbucket provides. During the review of the pull request a CI pipeline will be automatically started to validate the new changes does compile without warnings/errors, linting and acceptance testing.

5.3 Risk Analysis

Type: Project
Risk: Data loss
Likelihood: Low
Impact: High
Action: Backups of the repository and Overleaf documents are created using Git and stored on Github and BitBucket.

Type: Project
Risk: Unable to meet deadlines
Likelihood: Low
Impact: High
Action: Following our project plan will help reduce the chances of this happening. Weekly meetings with our supervisor will help us evaluate our progress.

Type: Project
Risk: Scope creep
Likelihood: Low
Impact: Medium
Action: To prevent scope creep we will be following the project plan. The requirements needs to be agreed upon in detail with the client before implementation starts.

Type: General
Risk: Client unavailable for longer periods
Likelihood: Low
Impact: Medium
Action: Contact supervisor and explain the situation.

Type: General
Risk: Sickness in group
Likelihood: Medium
Impact: Low
Action: The affected member will have to notify the group if a sickness will cause delay in their

scheduled work or if they are unable to attend meetings. The workload will in this case be split between the other members.

Type: General
Risk: Monthly pipeline build minutes runs out
Likelihood: Low
Impact: Low
Action: By setting the pipeline only to run on pull requests and set a reasonable time limit to avoid broken pipelines that deadlock all the minutes, this way we can reduce the build minutes used. If we do run out, we will have to run the test pipeline manually.

Type: General
Risk: Disagreement between members
Likelihood: Low
Impact: Low
Action: Project leader calls in disagreeing parties to meeting to discuss how to solve the issue. The supervisor can be invited to help us resolve the issue.

6 Project Plan

6.1 Work Breakdown Structure

Our work down structure consists of three sections; research, thesis and cockpit module for Suricata. Before we start we have to do research that will help us in implementing the plugin.We need to figure out the tools needed for collaboration and development, and how to set up the development environment. Then do research on how to write a good bachelor thesis by reading other students bachelor theses.

The second section is about writing the thesis itself. We have to submit a draft before the final submission. Therefore, it is important that we write enough in the first draft so we can get good feedback from our supervisor. This will help us improve the thesis before the final submission.

Last section focuses on setting up the development environment. This includes setting up the pipeline on Bitbucket, infrastructure on Openstack with a heat template as well as our local environment. Finally, we have to develop and test the plugin.

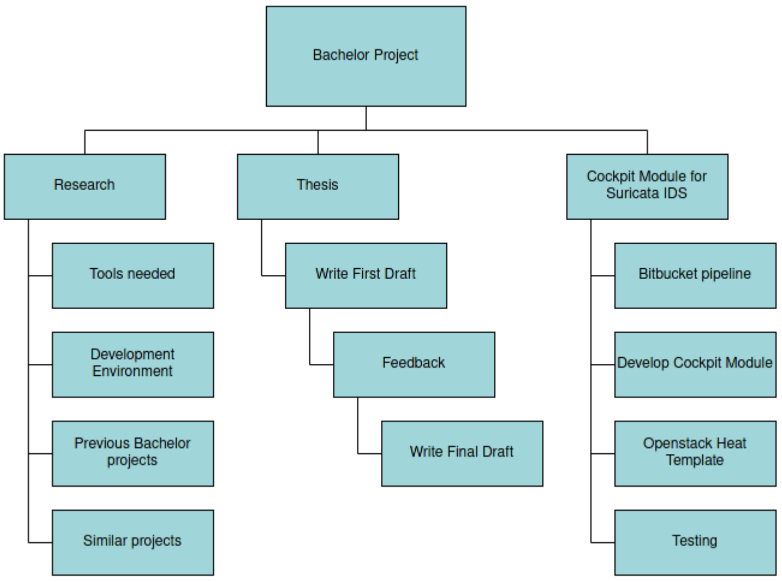


Figure 1: Work Breakdown Structure

6.2 Module architecture

The module consists of a web page with four tabs namely Service, Signatures, Logs, and Alarms. The Service page will present three buttons, start, stop and restart. This same tab will also display the state of the service.The next tab, Signatures, will be used to manage signature sources and update rule-sets. The Logs tab will show relevant service logs from the Suricata service and the Alarms tab will display alerts issued by Suricata. Signatures, Logs and Alarms tabs will have functionality for sorting by a relevant "field name".

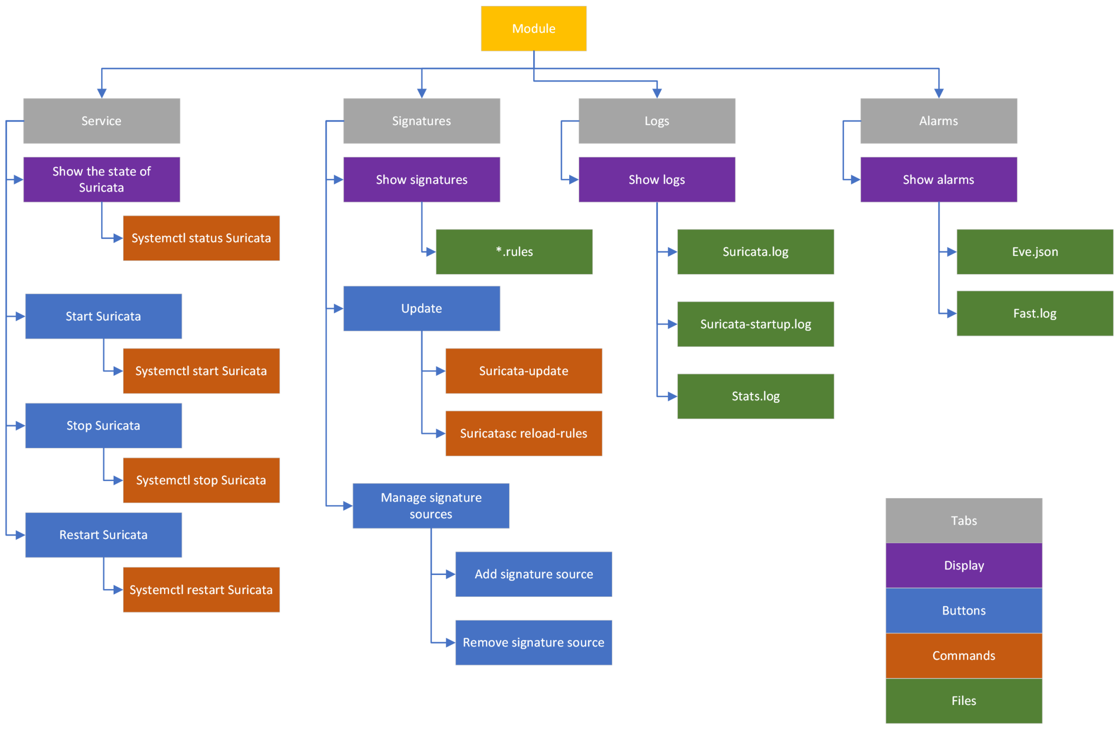
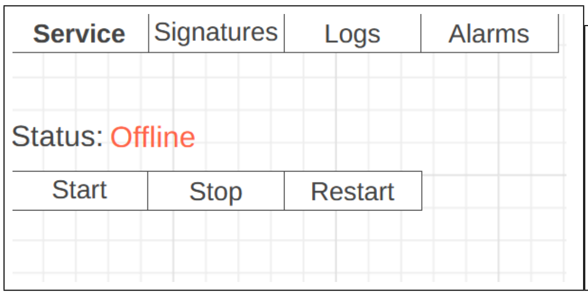


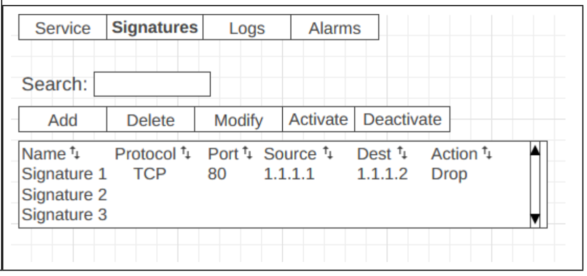
Figure 2: Module architecture

6.3 Interface mock-up

The following figures are our first drafts of the user interface mock-up. It is subject to change, and will not be an accurate representation of the final module.

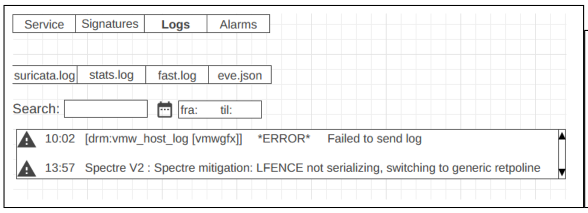


(a) Service



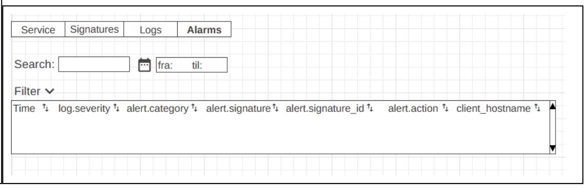
Name	Protocol	Port	Source	Dest	Action
Signature 1	TCP	80	1.1.1.1	1.1.1.2	Drop
Signature 2					
Signature 3					

(b) Signatures



```
10:02   [drm:vmw_host_log [vmwgfx]]   *ERROR*   Failed to send log
13:57   Spectre V2 : Spectre mitigation: LFENCE not serializing, switching to generic retpoline
```

(c) Logs



Time	log.severity	alert.category	alert.signature	alert.signature_id	alert.action	client_hostname

(d) Alarms

Figure 3: Mock up of the module

6.4 Overall system architecture

Cockpit and Suricata are installed on the same Linux machine. systemd services requests the status of the Suricata service and starts, stops and restarts Suricata. Cockpit will use the Unix socket to reload the signature rules when it's been updated. Information to display in the Cockpit module is extracted from the files generated by Suricata, these are represented by the green boxes on figure 4. Suricata uses the "suricata.yaml" file for configuration settings and Cockpit will also read the "yaml" file to understand where the rules and logs are located.

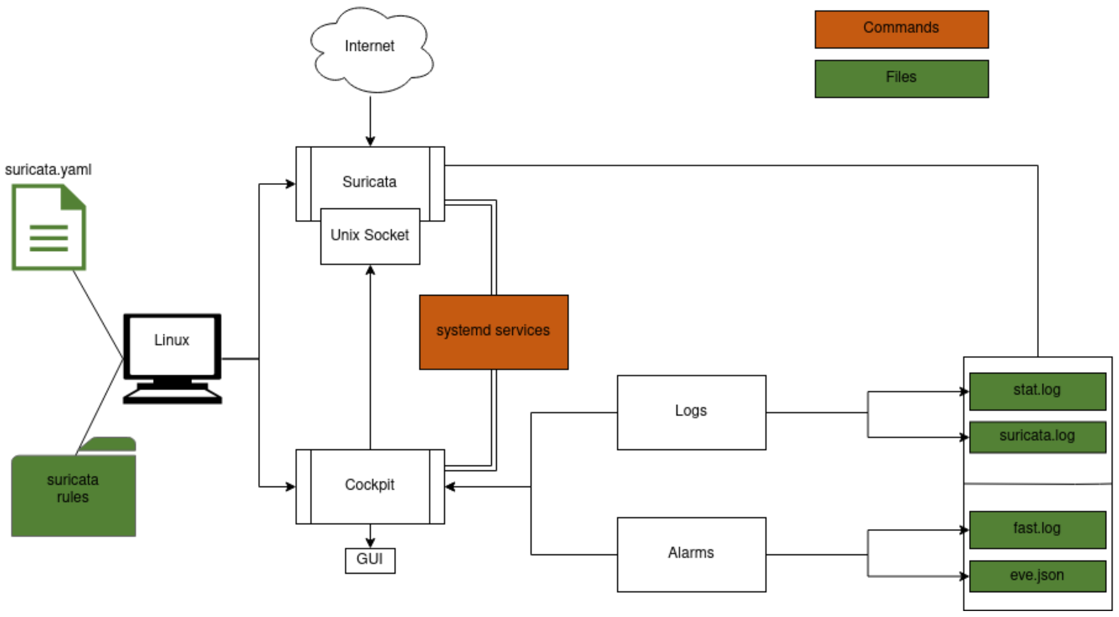
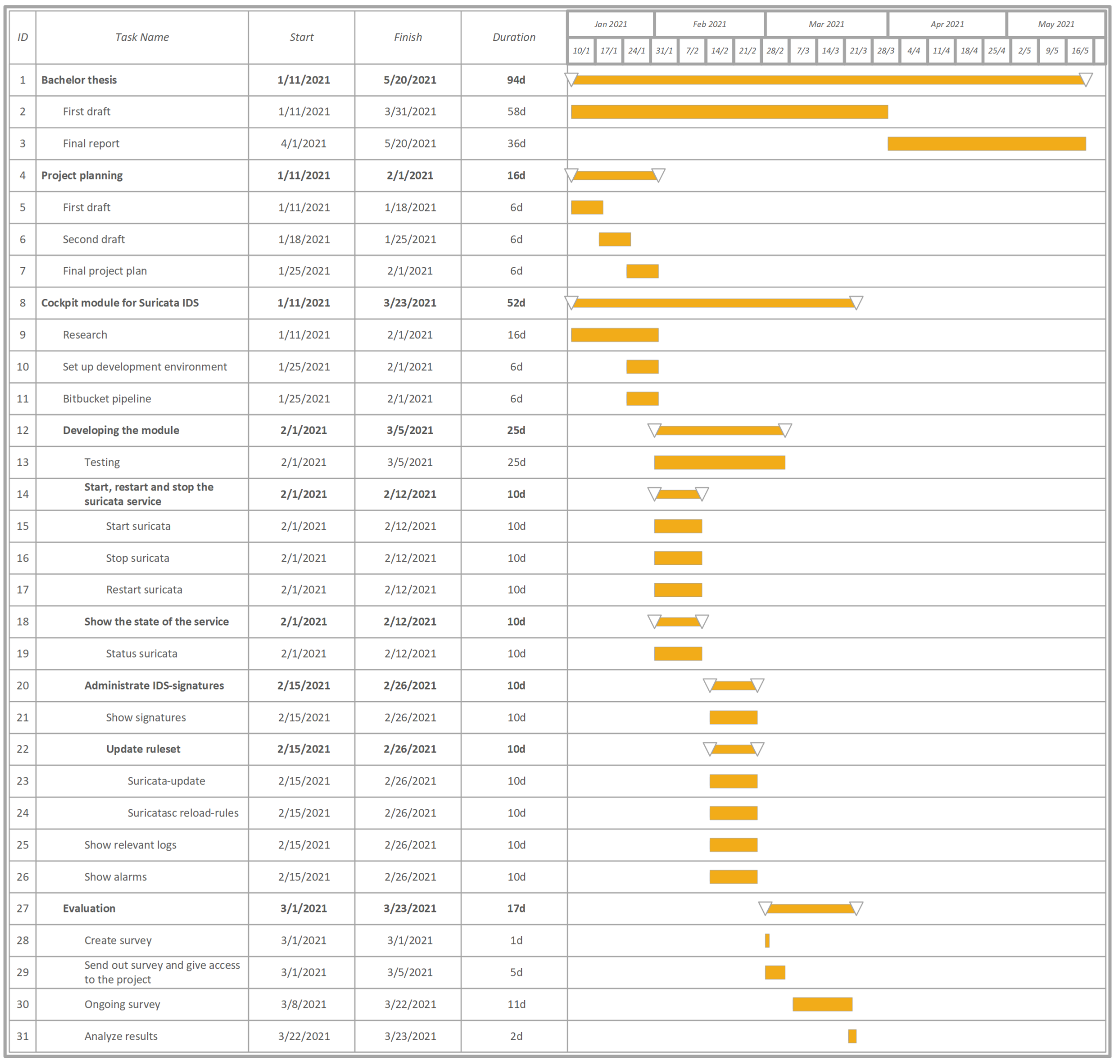


Figure 4: Overall System Architecture

6.5 Gantt Diagram



ID	Task Name	Start	Finish	Duration
1	**Bachelor thesis**	**1/11/2021**	**5/20/2021**	**94d**
2	First draft	1/11/2021	3/31/2021	58d
3	Final report	4/1/2021	5/20/2021	36d
4	**Project planning**	**1/11/2021**	**2/1/2021**	**16d**
5	First draft	1/11/2021	1/18/2021	6d
6	Second draft	1/18/2021	1/25/2021	6d
7	Final project plan	1/25/2021	2/1/2021	6d
8	**Cockpit module for Suricata IDS**	**1/11/2021**	**3/23/2021**	**52d**
9	Research	1/11/2021	2/1/2021	16d
10	Set up development environment	1/25/2021	2/1/2021	6d
11	Bitbucket pipeline	1/25/2021	2/1/2021	6d
12	**Developing the module**	**2/1/2021**	**3/5/2021**	**25d**
13	Testing	2/1/2021	3/5/2021	25d
14	**Start, restart and stop the suricata service**	**2/1/2021**	**2/12/2021**	**10d**
15	Start suricata	2/1/2021	2/12/2021	10d
16	Stop suricata	2/1/2021	2/12/2021	10d
17	Restart suricata	2/1/2021	2/12/2021	10d
18	**Show the state of the service**	**2/1/2021**	**2/12/2021**	**10d**
19	Status suricata	2/1/2021	2/12/2021	10d
20	**Administrate IDS-signatures**	**2/15/2021**	**2/26/2021**	**10d**
21	Show signatures	2/15/2021	2/26/2021	10d
22	**Update ruleset**	**2/15/2021**	**2/26/2021**	**10d**
23	Suricata-update	2/15/2021	2/26/2021	10d
24	Suricatasc reload-rules	2/15/2021	2/26/2021	10d
25	Show relevant logs	2/15/2021	2/26/2021	10d
26	Show alarms	2/15/2021	2/26/2021	10d
27	**Evaluation**	**3/1/2021**	**3/23/2021**	**17d**
28	Create survey	3/1/2021	3/1/2021	1d
29	Send out survey and give access to the project	3/1/2021	3/5/2021	5d
30	Ongoing survey	3/8/2021	3/22/2021	11d
31	Analyze results	3/22/2021	3/23/2021	2d

Figure 5: Gantt chart

6.6 Deadlines

- **Group agreement:** February 1st
- **Project Plan:** February 1st
- **First draft thesis:** March 31st
- **Thesis:** May 20th

6.7 Deliveries

- **Function: Start, restart, stop suricata service:** February 12th
- **Function: Show state of service:** February 12th
- **Function: Administrate IDS-signatures:** February 26th
- **Function: Show relevant logs:** February 26th
- **Function: Show alarms:** February 26th

References

[1] Cockpit-project, 2021. https://cockpit-project.org/.

[2] Wikipedia contributors. Intrusion detection system, 2021. https://en.wikipedia.org/w/index.php?title=Intrusion_detection_system&oldid=998288803.

[3] cockpit project. Running cockpit, 2020. https://cockpit-project.org/running.html.

[4] freedesktop. systemd system and service manager, 2020. https://www.freedesktop.org/wiki/Software/systemd/.

[5] Dan Radigan. What is kanban?, 2021. https://www.atlassian.com/agile/kanban.

[6] Alexander Sergeev. Tests in extreme programming, 2016. https://hygger.io/blog/tests-in-extreme-programming/.

[7] 2021. https://javascript.info/comments#good-comments.

[8] Ivar Farup. thesis-ntnu, 2020. https://github.com/COPCSE-NTNU/thesis-NTNU.

[9] Martin Pitt. Starter kit - the turn-key template for your own pages, 2018. https://cockpit-project.org/blog/cockpit-starter-kit.html.

A.4 Project Agreement

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

NTNU/SOC

____________________ (oppdragsgiver), og

Said-Emin Evmurzajev, Sigve Sudland

Sindre Morvik, Anders Svarverud

____________________ (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01.21 til 20.05.21.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle beståtte bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Jia-Chun Lin

Oppdragsgivers kontaktperson (navn): Christoffer Hallstensen

Student(er) (signatur): [illegible] dato 13.01.21

Sigve Sudmand dato 13.01.21

Anders Svalverud dato 13.01.21

Sindre Morvik dato 13.01.21

Oppdragsgiver (signatur): [illegible] dato 28.1.2021

NTNU
NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

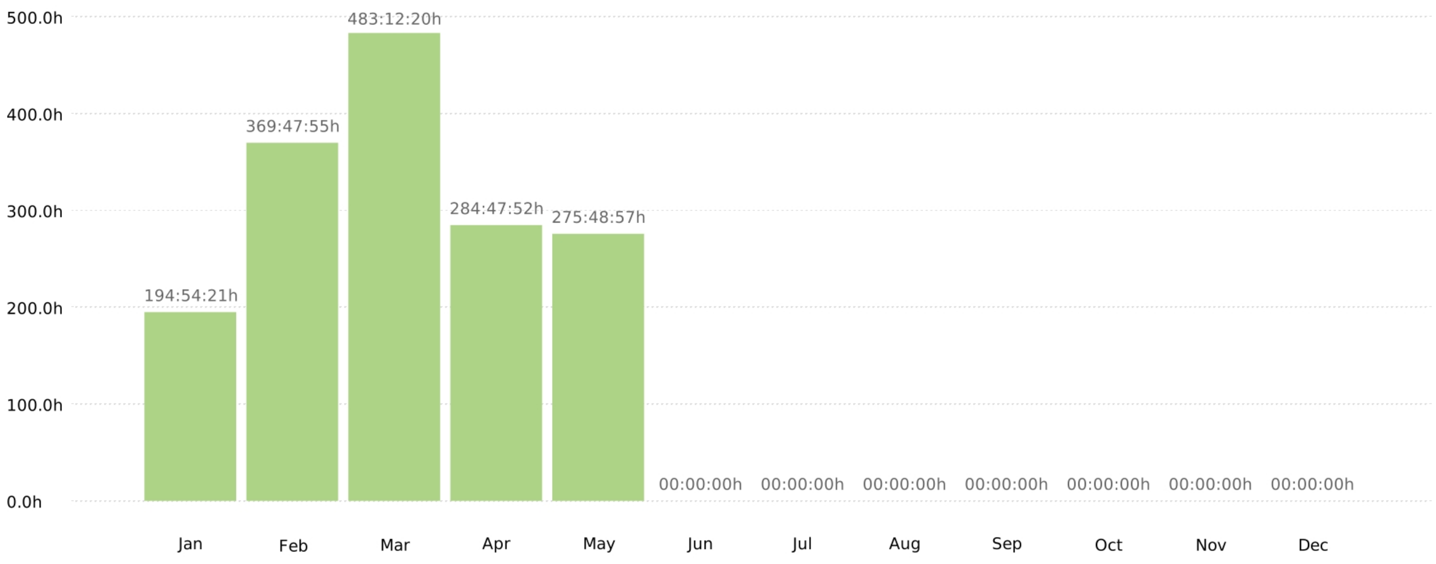
Instituttleder/faggruppeleder (signatur): ______________________ dato __________

A.5 Time log

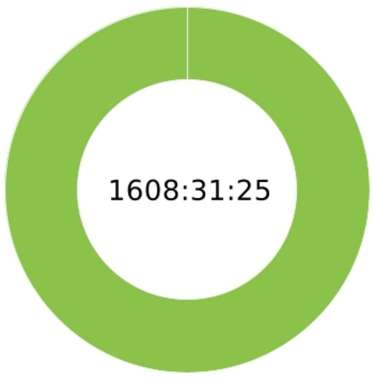
Summary report

01/01/2021 - 12/31/2021

Total: 1608:31:25 Billable: 00:00:00 Amount: 0.00 USD

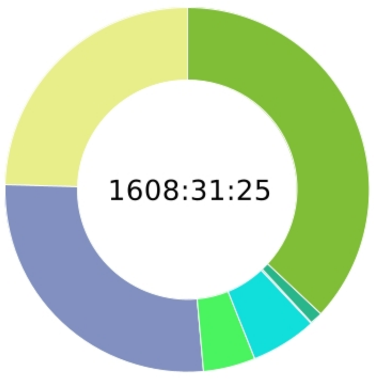


Project



●	Bachelor	1608:31:25	100.00%

Task



●	(Without Task)	395:19:21	24.58%
●	Development - Bachelor	431:44:45	26.84%
●	Generelt(møter, mails, kontaktpersoner) - Bachelor	75:05:26	4.67%
●	Prosjektplan - Bachelor	93:07:45	5.79%
●	Read other bachelor thesis' - Bachelor	01:30:26	0.09%

●	Research - Bachelor	18:10:16	1.13%
●	Thesis - Bachelor	593:33:26	36.90%

Project / Task	Duration	Amount
Bachelor	1608:31:25	0.00 USD
(Without Task)	395:19:21	0.00 USD
Development	431:44:45	0.00 USD
Generelt(møter, mails, kontaktpersoner)	75:05:26	0.00 USD
Prosjektplan	93:07:45	0.00 USD
Read other bachelor thesis'	01:30:26	0.00 USD
Research	18:10:16	0.00 USD
Thesis	593:33:26	0.00 USD

A.6 Link to module on Github

The module for Cockpit is open source and available on Github from the link provided here: `https://github.com/Sudland/cockpit-suricata/`

Cockpit module for administration of Suricata IDS

NTNU
Kunnskap for en bedre verden