GJØVIK UNIVERSITY COLLEGE

DEPARTMENT OF COMPUTER SCIENCE
AND MEDIA TECHNOLOGY

BACHELOR THESIS

Snowman

A complete, effective and secure rule management system for Snort

Authors:
Thomas NYHEIM
Eirik SKOGSTAD
Eigil OBRESTAD

Supervisor:
Slobodan PETROVIC

May 19, 2014

Participants

Thomas NYHEIM, thomas.nyheim@hig.no
Eirik SKOGSTAD, eirik.skogstad@hig.no
Eigil OBRESTAD, eigil.obrestad@hig.no

Supervisor

Slobodan PETROVIC, slobodan.petrovic@hig.no

Employer

CENTRE FOR PROTECTION OF CRITICAL INFRASTRUCTURE
Cyber Services and Operations
Norwegian Armed Forces Cyber Defence

AVDELING FOR BESKYTTELSE AV KRITISK INFRASTRUKTUR
Cyber Tjenester og Operasjoner
Cyberforsvaret

Employer Contact

Capt. Jarle KITTILSEN, jkittilsen@mil.no, +47 6110 3850

Keywords

Software Development, Intrusion Detection System, Software Performance Testing, Snort

80 Pages - **9** Appendices

AVDELING FOR
INFORMATIKK OG MEDIETEKNIKK
HØGSKOLEN I GJØVIK
POSTBOKS 191
2802 GJØVIK

DEPARTMENT OF COMPUTER SCIENCE
AND MEDIA TECHNOLOGY
GJØVIK UNIVERSITY COLLEGE
BOX 191
N-2802 GJØVIK
NORWAY

English Abstract

In today's technological world, the Internet has become an integral part of businesses and organisations. However, the Internet brings a large amount of threats against computer networks, as individuals and organisations seek to exploit them for monetary gain, information superiority or political and social activism. In order to properly protect a computer network, it is vital to be able to detect these threats, which is commonly done by using an intrusion detection system (IDS). An IDS will normally operate by analysing the network traffic by comparing it to threat-patterns defined in signatures, and raising alerts when threats are detected. Given the number of possible threats to a computer network, an IDS often has a large amount of signatures that must be properly managed in order to maintain both a secure and usable network. Signature management therefore involves enabling and disabling the signatures, as well as keeping them up to date. This can be a challenging task for even small-sized organisations, as the IDS might operate on multiple parts of the network simultaneously.

This thesis presents a software which facilitates signature management in environments with multiple instances of the IDS, called sensors, by using a central server which automatically updates and distributes signatures to the sensors. Furthermore, the software introduces a novel way of automatic management of signatures on each sensor. The software is controlled with a simple and intuitive graphical user interface which supports multiple users. This thesis also compares the functionality and efficiency of the software with similar signature management systems through a theoretical analysis of performance, complemented with practical tests. The testing has also demonstrated the importance of conducting performance tests during software development, in order to identify and improve inefficient code in the software.

Norwegian Abstract

Dagens samfunn er helt avhengig av internett for kommunikasjon og samhandling. Dette gjelder ikke minst for bedrifter og organisasjoner, hvor internett blandt annet brukes til å nå ut til, og kommunisere med, kunder og partnere. Et stort problem med bruken av internett er at det gjør datamaskiner og interne datanettverk sårbare for eksterne angrep, da disse kan inneholde, eller gi tilgang til, sensitiv informasjon som kan utnyttes av organisasjoner og enkeltpersoner med hensikter som for eksempel økonomisk vinning. Oppdagelse av potensielle angrep er avgjørende for å best mulig kunne beskytte datanettverk, noe som normalt gjøres ved hjelp av et såkalt *intrusion detection system* (IDS). Et IDS opererer vanligvis ved å overvåke og analysere nettverkstrafikk, som sammenlignes med mønstre definert i signaturer og sender en advarsel når en trussel blir gjenkjent. Mengden av potensielle angrep som kan utføres er enorm, noe som gjør at et IDS inneholder en stor mengde signaturer som må vedlikeholdes daglig for å kunne møte et trusselbilde i stadig endring. Vedlikeholdet av signaturer kan være krevende selv for små bedrifter, da IDS-et vanligvis opererer flere steder på datanettverket samtidig.

Denne rapporten presenterer en programvare for enkel og effektiv håndtering av signaturer for systemer med flere instanser, kalt sensorer, av et IDS. Dette gjøres ved å ha en sentral tjener som håndterer automatisk oppdatering og distribuering av signaturer til sensorene. Programvaren inneholder også et klient-program som kjører på hver sensor og muliggjør en effektiv og automatisk håndtering av signaturer på hver enkelt sensor. Programvaren styres av et enkelt og intuitivt grafisk brukergrensesnitt med støtte for flere brukere. Rapporten inneholder også en studie av programmets ytelse sammenlignet med lignende programvare, i form av en teoretisk og praktisk analyse. Den praktiske analysen gir i tillegg et innblikk i hvor viktig ytelsestesting under programvareutvikling er for å identifisere og forbedre ineffektiv kode i programvaren.

Preface

Acknowledgements

We would like to thank our supervisor from Gjøvik University College, prof. Slobodan Petrovic, for his help and input during the project. We would also like to thank Avdeling BKI for assigning us this project and the help with forming requirements and the software testing, and especially Jarle Kittilsen who took on a role as a supplementary supervisor.

We would also like to acknowledge Gjøvik University College, that has been an arena for expanding our knowledge over the past few years and the knowledge we've accrued are the foundation of which this project is laid upon.

About this document

This document is written in LaTeX and is based on a template written by Ivar Farup, Kjetil Orbekk and Simon McCallum for master theses at Gjøvik University College. The bibliography is produced with BibTex and follows the Vancouver standard for citations. The bibliography style definition is provided by Folkert van der Beek.

Legal

Logo artwork commissioned by Kerrigan, `http://rattlesire.deviantart.com/`.

Glossary

This chapter contains an alphabetical list of technical terms used throughout the report along with their definition.

AJAX Asynchronous JavaScript and XML.

API Application programming interface.

CLI Command-line interface.

CSS Cascading Style Sheets, a markup language for web pages.

Daemon A computer program that runs as a background process, usually without any user interaction.

DOM Document object model.

HDD Hard disk drive.

HTML Hypertext markup language.

HTTP Hypertext transfer protocol.

I/O Input / Output, used to describe data flow to and from devices such as hard drives.

IDS Intrusion detection system.

MD5 A cryptographic hash function producing a 128-bit hash value.

MVC Model-view-controller.

Open Source Software Software where the source code is publicly available and is licensed in such a way that the copyright holder grants the rights to study, alter and redistribute the software to anyone for any purpose [1].

ORM Object-relational mapping.

RAM Random access memory.

RPC Remote procedure call.

RPM Rounds per minute.

SCP Secure Copy. An encrypted file transfer protocol.

Sensor A computer system wherein Snort and/or other security software is recording network traffic and produces alerts when malicious content is detected.

SSD Solid state drive.

SSL Secure sockets layer.

TRIM Garbage collection routines for SSD.

URL Uniform resource locator.

WSGI Web Server Gateway Interface.

XML Extensible markup language.

Contents

List of Figures

List of Tables

1 INTRODUCTION

1.1 Project Background

The Norwegian Armed Forces Cyber Defense unit for Computer Network Defense (CND), Centre for Protection of Critical Infrastructure (Avdeling BKI in Norwegian), is responsible for detecting and stopping cyber-attacks against the critical infrastructure and command and control systems utilized by the Norwegian military. One of the tools used to carry out this task is the open source IDS Snort [3].

Snort is a leading [4] open source IDS that is used to detect potentially unwanted or malicious network traffic based on predefined patterns described in signatures. There are both free and commercial signatures available for Snort and as Snort has been highly adopted by the security community, these signatures are continually updated and maintained. For large networks, an IDS will often operate in multiple parts of the network simultaneously, where each separate instance of the IDS is known as a sensor.

The management of Snort signatures (henceforth referred to as rules) is cumbersome in multi-sensor environments without a centralized service which can assure that all sensors are synchronised with the latest rules. Avdeling BKI has therefore challenged a group of students from Gjøvik University College (GUC) to design and develop a functional, effective and secure rule management system for Snort that can continue to live on beyond this project.

1.2 Previous Work

Studies [5] have shown that a good rule management system for Snort will improve the workflow and efficiency of the IDS significantly, and a prototype for such a system was built in 2012 called Bring Home The Bacon [5] (henceforth referred to as BHTB). This prototype has been in use at Avdeling BKI since 2012, but is suffering from lacks of functionality and efficiency due to being a proof-of-concept prototype. BHTB has, however, proven that there is a significant gain from having an intuitive Graphical User Interface (GUI) for managing rules. A GUI better facilitates the work of the operator, as work is shifted from complex operations in a CLI to automatic execution of common routines triggered by buttons.

There was another similar project conducted at Gjøvik University College in 2012 called Snortmanager [6], which also deals with management of Snort rules. The Snortmanager project files are available as open source software, but no usage instructions are available. The source code has not been maintained since its initial release and is designed for a specific company. Enabling, disabling and configuring rules is also cumbersome in this system as it is based on plaintext arguments in "policies" that users must construct themselves [6]. The rule distribution is also done manually from the files generated. All this makes Snortmanager unsuitable for Avdeling BKI and their routines.

Several add-ons and other Snort-related software exist [7], and a few of these are built to manage Snort rules, most notably Pulled_Pork [8] and Oinkmaster [9]. The key differences between all software mentioned in this section and Snowman are further explained

in Chapter 2.

1.3 Project Description

The purpose of this project is to design a system that provides a simple and efficient way of managing rules for the Snort IDS. The project is a successor to BHTB and expands on the good ideas from BHTB, and adds functionality requested by Avdeling BKI to create a more adoptable open source rule management system for Snort. The project can be broken down into four main objectives:

1. Identify the challenges of rule management in an IDS such as Snort.
2. Describe how a good rule management system should operate.
3. Develop a good open source rule management system for Snort, at a high technology readiness level (TRL) [10].
4. Measure the performance of the system and compare this to other similar systems.

The first two objectives were addressed early on in the project during the initial design phase of the software. To develop a good rule management system, it is essential to consider the tasks in the two first objectives, as they form the basis of how the system in the third objective is designed. Challenges and best practices in rule management is therefore presented throughout this report to support the design decisions made in the final software solution.

As the project became very performance-oriented, the report also contains an initial study of software performance and practical performance test of related software.

1.4 Target Audence

Snowman is targeted at users of the Snort IDS in a larger production environment with many sensors and networks. More specifically, the target audience consists of security experts with a high degree of knowledge of Snort and computers in general.

This report is targeted at those who are interested in discovering the complexity of the rule structure of Snort, and academics that are interested in reading about IDS software performance testing.

1.5 Project Objectives

1.5.1 Result Objectives

As stated in the project plan (Appendix A), this project has a goal of researching, designing and developing a system that as a minimum:

- Is more efficient than BHTB.
- Is developed as an engine that responds to API-calls.
- Is modular in design and allows future changes and additions that will not affect existing functionality.
- Is scalable enough to handle both small and large sets of IDS-sensors and rules in a production environment.

- Is able to download rules and rulesets from multiple customizable sources.
- Is able to distribute different rules to different IDS-sensors.
- Can download and distribute rules automatically at set intervals.
- Is capable of exporting rules.
- Has the capability to activate, deactivate, filter and suppress rules on a per sensor basis, in a user friendly way.
- Supports multi-user interaction.
- Can efficiently and safely work with different installations of Snort and various other third party tools, such as SNORBY [11].
- Is easy to install.
- Has an internal logging system.
- Is well-documented and can be easily adopted by anyone as open source software.

In addition, the project group will attempt to add the following functionality:

- A graphical user interface.
- The mentioned user interface must be user friendly and intuitive, and require only limited knowledge of Snort rules from the user.
- The system can also distribute rules to Suricata [12], another Snort-like IDS, which can operate with the exact same rules as Snort.
- Implement security features that ensure confidentiality of rules.
- The project will investigate and potentially implement multi-threading technology or other optimization techniques to exploit modern hardware capacities.
- The system is capable of validating and checking rules for errors before they are distributed to sensors.
- The system supports commenting of the rules.
- The system supports writing custom rules and editing existing rules.

Once the system is developed, it will be released as an Open Source Software to the public in a well-documented format.

1.5.2 Effect Objectives

Forming baselines from the current prototype BHTB and from Snortmanager, the project has a goal to further increase the efficiency of rule management by investigating and implementing programming techniques such as multi-threading and caching to reduce latency and delays in the system by as much as 50%. The software must also maintain sublinearity in performance, i.e. 10 executions of the same job should take 10 times, or less, the amount of time as a single execution.

The achievement of these goals is to be measured by conducting several performance tests on the final software solution. Results will also be compared to the performance of BHTB and Snortmanager.

1.6 Academic Background

This project has benefited greatly from the fact that all three project members are all taking different computer science courses. Eirik is currently completing his undergraduate engineering degree in computer science at Gjøvik University College, and has practical experience in software development from a one-year technical work internship at CERN in Switzerland. Eigil is completing his undergraduate degree in network and system administration at Gjøvik University College, and has also spent one year at CERN working on network monitoring and administration. Thomas has been studying information security at Gjøvik University College, and has worked for Avdeling BKI for the past seven years prior to this project. He has been working with threat detection and network traffic analysis, and has good knowledge of what Avdeling BKI needs, in addition to general experience from the information security field. Given the variety of knowledge from each members background, the project group has enjoyed a unique opportunity to draw from each other's strengths, and approach the project from different perspectives.

1.7 Framework

1.7.1 Software Development Methodology

As this project has been very time restricted, the development needed to be as efficient as possible. The project group therefore had to find a model that utilized the available manpower in the best possible way. Considering the small size of the project group, large and complex development models would add too much bureaucracy to coordinate the project. A lightweight and efficient development model was preferred, so the group chose to utilize an agile development model [13] (Figure 1) structured into development cycles. This way, each group member could take an active part in the project and all members could feel a sense of ownership.

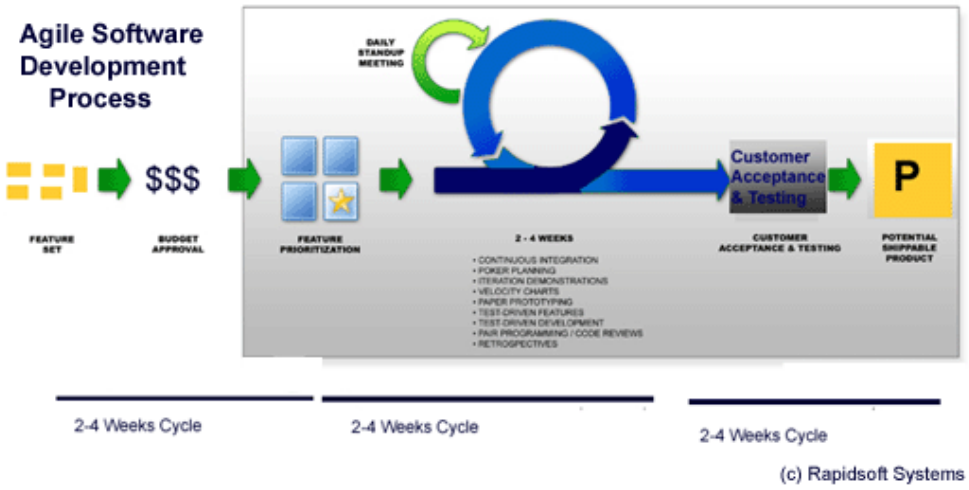


Figure 1: Agile development model [2]

By developing iteratively and incrementally, the group combined the best of both the agile development world and from the very well-defined waterfall approach [14]. Feedback from Avdeling BKI and the project supervisor has been very important in the course of the project, in addition to having a structured and reasonable way of doing

changes to the plan while the project was running. At the same time the group was able to maintain the overall plan within cycles, so that it was possible to work individually, while still being well-coordinated.

To be able to develop all the modules of the project within the time frame, each development cycle consisted of multiple modules or functionality developed in parallel. This allowed for good utilization of time and personnel. The start of each cycle consisted of a meeting where the goals for the cycle were defined and individual tasks that we needed to get done were identified. This process bears similarity to the start of a sprint in the development model Scrum [15].

During the development part of the cycle, the group worked individually on the tasks that were defined in the planning phase. Regular meetings and cooperative programming within the group ensured that everyone pulled the project in the same direction. At the end of each cycle a working set of software was ready, which ideally complied with all the goals set at the start of each cycle. Meetings with Avdeling BKI were held between each cycle to present the progress, and to let them influence the direction of the project.

When all the development cycles were completed, final assurances were made that everything worked as intended and that all modules coexisted in harmony. This part of the development also consisted of some hands-on testing by Avdeling BKI and performance testing.

1.7.2 Schedule

This project had a comprehensive initial phase for mapping and designing of the system. The group spent the major part of the first two weeks planning how to implement the system. After the two weeks and before the start of the development, a “project decision meeting" was held with Avdeling BKI to receive feedback on the design and confirm that the system was modelled correctly for their needs.

After this, the implementation phase commenced, which consisted of development cycles as described above in Section 1.7.1. This phase was divided into five parts: four development cycles and a final cleanup/testing cycle. All the development cycles lasted two weeks, aside from the last development cycle and the cleanup/testing cycle, which lasted one week each. During these cycles the group conducted meetings with both the project mentor and Avdeling BKI every two weeks, timed so that these meetings took place on alternating weeks. This ensured continual feedback and advice throughout the development process.

After the development phase, focus was shifted from the software to the project report, and the group spent the final 5-6 weeks working on the report. The group benefitted from the work performed during the first weeks of the project which could go almost directly into the report.

1.7.3 Project Organization

Reflecting the fact that the project had multiple phases, the organisation of the project changed throughout the project. During the first few weeks the group spent most of the time together in a group room, cooperating in forming the design and discussing various solutions. Once most of the details had been worked out and the implementation phase began, there was more room and need to work individually to produce code. In this period, the group gathered together a few times a week to coordinate the progress and plan the next course of action. Throughout the project, most decisions have been made

either by speaking with Avdeling BKI or achieving consensus within the group.

1.8 Document Structure

INTRODUCTION - Contains an overview and description of the project and its goals.

EXISTING SOLUTIONS - Describes similar software and information relevant to the project.

REQUIREMENTS SPECIFICATION - Describes the requirements and scope of the software development.

CONCEPTS OF SNORT - Explains the basic concepts of Snort.

DESIGN - Details the architecture and design of the system.

IMPLEMENTATION - Describes how the system has been implemented.

SOFTWARE TESTING - A short description of how the system has been tested and some of the practical problems the project has encountered.

PERFORMANCE ANALYSIS - Describes what and how we have tested the system for performance.

RESULTS OF PERFORMANCE TESTING - Lists the results of the performance testing.

ANALYSIS OF PERFORMANCE TESTING RESULTS - Contains an analysis and concludes the findings of the performance tests.

CONCLUSIONS - Summarizes the findings in this report.

GROUP EVALUATION - The group evaluation of this project.

BIBLIOGRAPHY

APPENDICES - Some parts of the appendices are only available in Norwegian.

1.8.1 Special Styles Used

Software names are written in a different font throughout the report, like so: Snowman. Source code is listed in an enclosed box with syntax highlighting (Listing 1.1). Source code can also appear in the text with the same syntax highlighting as the listings, especially note names of modules and classes which will appear as so: `module`, `Class`.

```
1 from module import Class
2
3 class Hello:
4   def printHello():
5     # Print a message
6     print 'Hello World!'
```

Listing 1.1: Source code example.

2 EXISTING SOLUTIONS

The following sections will quickly elaborate on the functionality of software similar to Snowman to identify the differences, weak and strong sides of the different solutions.

2.1 Bring Home The Bacon

Bring Home The Bacon was a by-product prototype of the Master thesis written by Henriksen in 2012 [5]. In the thesis, Henriksen seeks to prove that having a central, graphically-oriented system to administer multiple Snort-sensors and their rules, results in a signficant improvement in user workflow over the more traditional command-line environment. Henriksens tests show that such a system can double the users' work efficiency, if not more, as the command-line environment would scale very badly as the number of sensors increases. Henriksen also outlines a rule management process, which has been helpful during the initial phases of this project to map the program workflow (Section 3.1.1).

Despite being a prototype, BHTB offers the essential functionality required for managing rule sets on multiple sensors. It can display a list of rules which can also be turned on and off on different sensors, as well as a list of rulesets that can also be turned on and off, which will effect all rules in the ruleset. This is considered a drawback, as BHTB does not offer a mechanism where rule sets can be disabled without affecting the state of the rules within. This means that the original state of the rules in a ruleset is lost whenever the state of the rule set is changed. Snowman improves this by keeping the state of the rulesets separate from the state of the rules.

BHTB distributes rules to the sensors by generating all files centrally and sending them one by one via SCP. It determines which rules to distribute by comparing the main rule database table with a second table representing which rules the sensor should not have. This way of distributing rules is considered ineffective, and has been solved differently in Snowman (explained in Section 5.1). BHTB also supports manual creation of rule filters and suppressions, but does not offer any input validation, which makes the sensors vulnerable to crashes[1] in the case of invalid input. This weakness is taken into consideration in Snowman, which offers form validation.

BHTB is programmed with Python and uses the Tornado framework [16] as its webserver, which works similar to Django [17], a framework used by Snowman, with its URL API system (see Section 5.2.2). It also uses SQLite [18] as its database, which has limited abilities when it comes to relations and more complex data structures.

According to Avdeling BKI, BHTB really begins to show its limitations once the system is populated by a lot of rules[2] and sensors, which makes it clear that it has a scalability problem. Avdeling BKI has used BHTB in a production environment since 2012 and experiences the software as slow and cumbersome.

[1] Snort will crash and not restart if something in the configuration files is malformed.
[2] In the five digit range.

2.2 Snortmanager

Snortmanager was created for a Bachelor thesis written at Gjøvik University College in 2012 [6], where the main purpose of the software was to make rule management more effective for the employing company. This was done by creating a central server with a rule-database which automatically downloads and updates rules from external sources. The user must then manually input which configurations are needed for a specified sensor and the system then creates configuration files for the sensor. As Snortmanager has a high focus on producing configuration files containing the necessary rules and settings for each sensor, but does not automatically distribute the files to the sensors like Snowman does, the user is required to manually move the files to the sensor through some unrelated process.

2.3 Other Notable Programs

There are two other pieces of software that should also be mentioned. Pulled_Pork and Oinkmaster are both programs that can download rules and update Snort-sensors. These programs contain some of the features found in Snowman and BHTB, and Pulled_Pork even supports Shared Object rules [19], which is not part of this project due to complexity. However, both of these programs suffer from the fact that they are command-line oriented. They are both very good at what they do, but lack the usability and efficiency offered by GUIs with buttons. These programs are a good choice for rule management in a small environment with one or very few sensors, but once the environment grows larger, with tens or even hundreds of sensors, they become ineffective.

2.4 Research on Software Performance

Since this project also investigates software performance, it is relevant to take a look at what others in the field have written on the subject. This subject is missing from most software development reports from Gjøvik University College, which can be explained by the usual high focus on agile development. [20] mentions that agile programming is becoming mainstream and by its nature is a concern for the future of software performance engineering, as the primary focus of an agile project is on functionality.

A proper performance testing methodology was not immediately available for this project. However, the papers [21] and [22] gave a few pointers towards developing the test cases described in Chapter 8. Both articles also stress the importance of performance testing during the development rather than after it, as this allows for time to actually implement optimizations and improvements. The performance testing in this report does not strictly follow any of these methodologies, but it includes the most important elements of them, e.g. performance testing during development.

Early on in the project it became clear that Snowman is very I/O-bound, as it performs a lot of database operations. The tests in this report show a noticable difference in performance between traditional rotational hard drives and SSDs. Myers comments on this in his master thesis [23] which shows that SSD has a significant advantage over HDDs when it comes to random read performance but not when it comes to write performance. Note that these tests were performed in 2008, a time when SSD technology was fairly new and suffering from its different way of writing data[3]. As operating systems, drivers

[3] Features such as TRIM were introduced shortly after Myers thesis.

and disk controllers have improved, SSDs are now superior to HDDs in most tasks.

2.5 Functionality Comparison

To easily visualize how various rule managers differ from each other, we have gathered key functionality from the most important pieces of such software and created a list of support for that functionality, which can be seen in table 1.

Feature	Snowman	BHTB	Snortmanager	Pulled_Pork	Oinkmaster
Rule download and storage	✓	✓	✓	✓	✓
Sensor administration	✓	✓	✓		
Graphical user interface	✓	✓	✓		
Automatic updates	✓	✓		✓	✓
Can synchronise rules to sensors	✓	✓			✓
Automatic synchronisation	✓	✓			✓
Enable/disable rules	✓	✓			✓
Supports multiple sensors	✓	✓			✓
Displays change history of updates	✓	✓			✓
Can store meta-data about rules	✓	✓			
Can turn on or off rulesets	✓	✓			
Can turn on or off rulesets on individual sensors	✓	✓			
Can add filters and suppressions to rules	✓	✓			
Can add filters and suppressions to rules on a per sensor basis	✓	✓			
Has an internal logging system	✓	✓			
Can display list of rules in the system	✓	✓			
Can display meta-data in the system	✓	✓			
Normalized database	✓		✓	N/A	N/A
Does not require root to interact with Snort	✓				
Turning rulesets on or off is non-propagating	✓		N/A	N/A	N/A
Can import filters and suppression from rule files	✓		✓	✓	✓
Has input validation	✓			N/A	N/A
Supports multiple database technologies	✓			N/A	N/A
Handles multiline rules	✓				✓
Can skip certain files	✓		✓		✓
User can approve changes before sensors are updated	✓				✓
Is easy to install	✓				

Can import rules from arbitrary sources	✓				
Can organize rulesets in a hierarchy	✓				
Can create new custom rulesets	✓				
Can organize sensors in a hierarchy	✓				
Can syncronize rules to multiple sensors at a time	✓				
Has a user system	✓				
Supports autonomous sensors	✓				
Does not distribute everything every time	✓				
Supports SO rules				✓	✓
Can create new rules					
Number of objects in data structure	16	4	9	N/A	N/A

Table 1: Functionality comparison chart.

3 REQUIREMENTS SPECIFICATION

Many of the requirements for this system originates from functions that already exist in BHTB which need to be reproduced or improved. This means that the requirements for this project are closely tied to the way BHTB operates and its general workflow. This section outlines the specifications gathered from BHTB and from conversations with BKI.

3.1 Functional Requirements

3.1.1 Program Workflow

To aid the development of the functional requirements, the program workflow was charted in order to give a better understanding of what the system was expected to do. This was accomplished by going through the functionality of BHTB with Avdeling BKI and detailing the functions and various needs for improvement or additional functionality. From this, a workflow chart (Figure 2) was drawn with the major parts of the system. In addition, Henriksen's management process [5] was taken into consideration.

To explain the workflow chart in Figure 2: the blue boxes are the major system functionalities, the green boxes are the major goals of the system and the arrows show the direction of flow the system should have. As can be seen, all of the program flow ends up with rules being synchronised to the sensors, making that a very central part of the system. It is important to note that in the beginning of the development, we decided that writing rules, automatically check for errors and adding user comments to them were nice-to-have (but still important) features that we could implement if we had time to do so. Hence, they are greyed out in the drawings, and left for future work.

From the workflow we could outline a rough modular design (figure 3) based on the naturally separate functions of updating rules, tuning them, distributing them and showing them. We also added a database module, since most of our program was going to be centered around it.

Rule download module

This module will handle all functionality related to downloading rules, including comparing MD5 checksums.

Rule pre-storage processing module

This module will essentially take downloaded rules, and parse and process them into objects that can be fed into the database.

DB Module

This module will handle all input and output to/from the database.

Rule pre-distribution processing module

This module will essentially take rules stored in the database and prepare them for distribution to sensors.

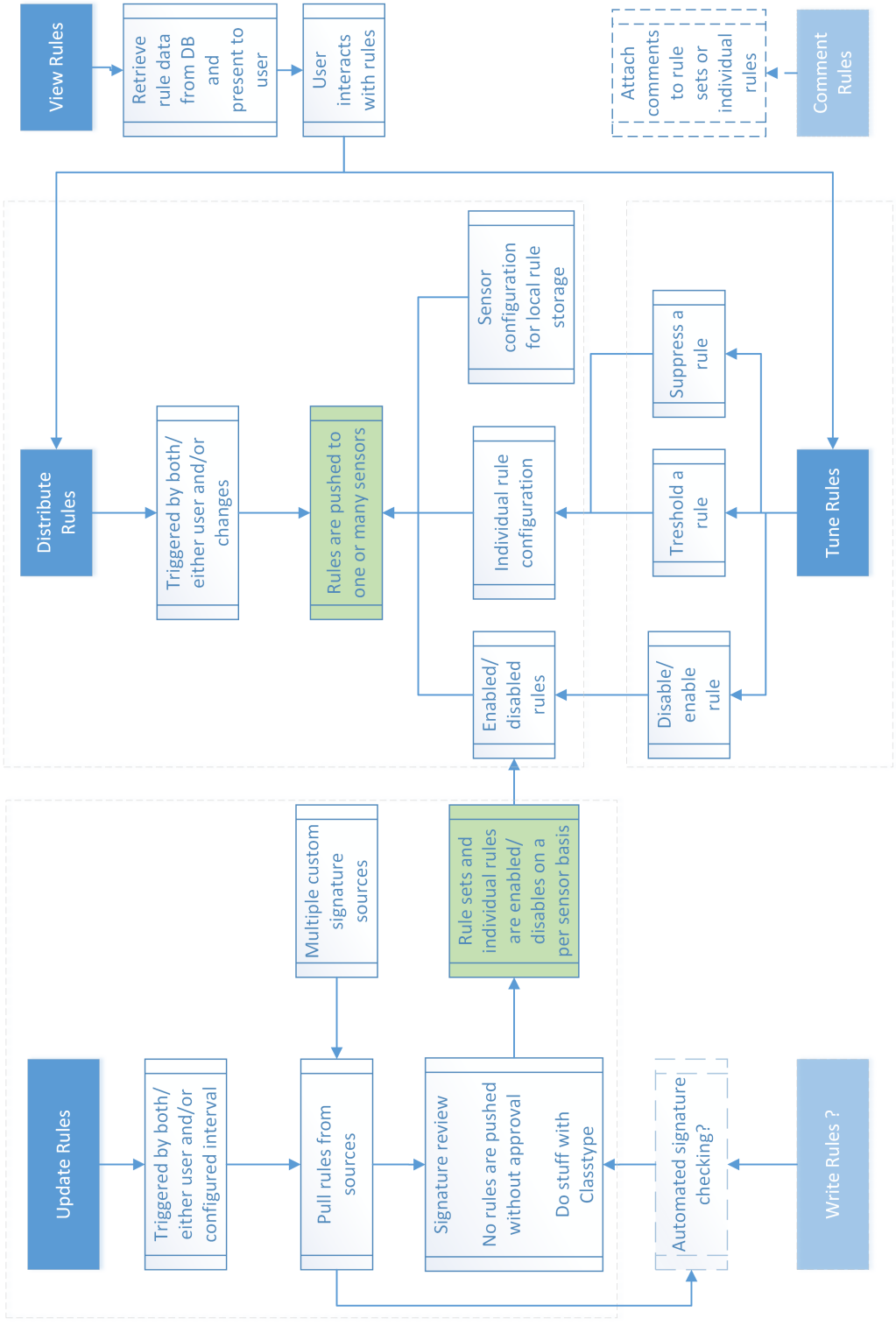


Figure 2: Program workflow

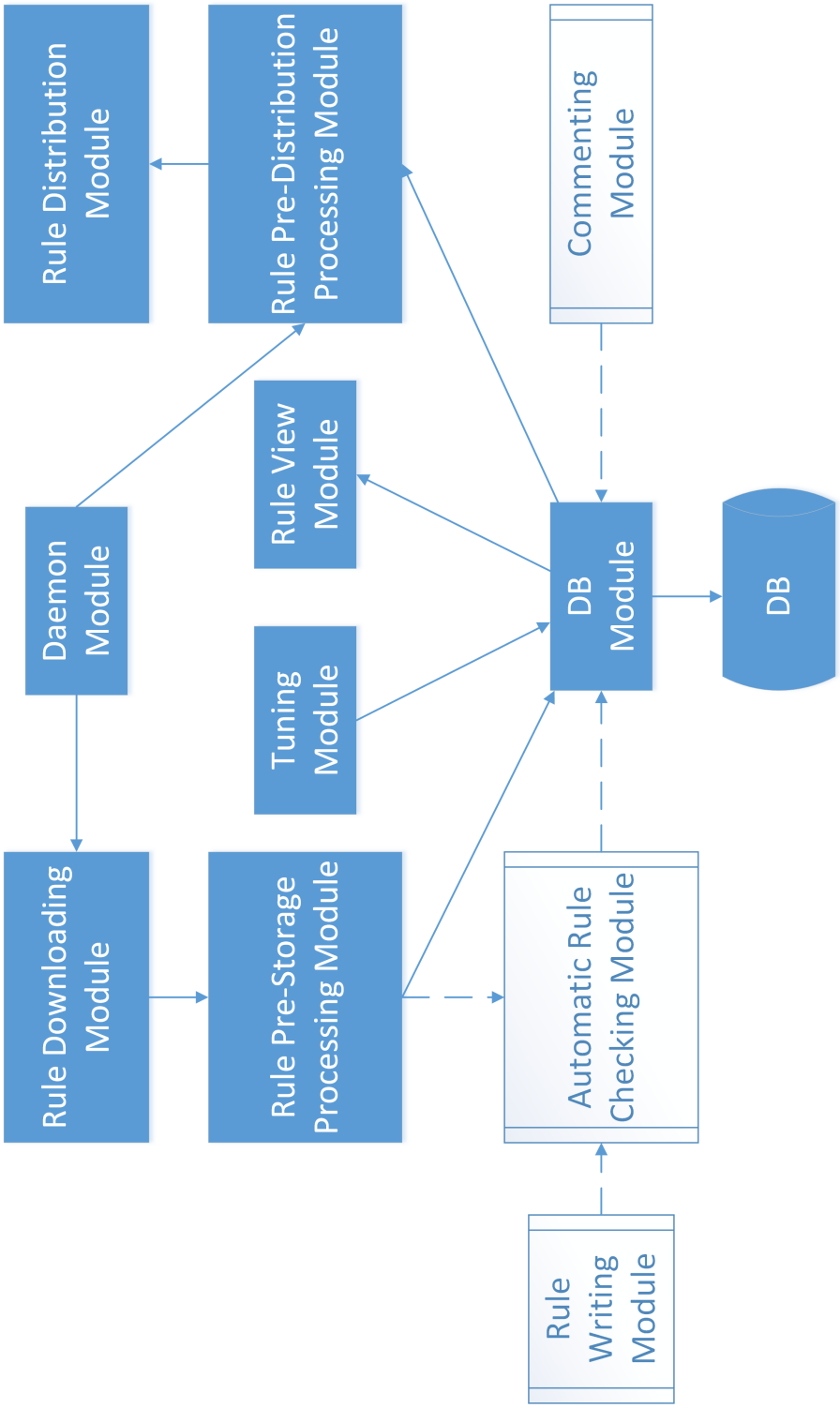


Figure 3: Program modules

Rule distribution module

This module will be in charge of distributing rules out to the Snort sensors and handing the rules over to Snort itself.

Tuning module

This module will handle all functionality related to turning rules on or off, filtering them or suppressing them.

Rule view module

This module will contain all the functionality needed to display the system status and system functions to the user. It will be the primary way the user interacts with the system, and thus contain the GUI.

Daemon module

This module will run continuously on the system to ensure that the system is responsive and to carry out any timed tasks, such as automatic updates or automatic distributions.

Commenting module

This module will allow users to attach comments to rules.

Rule writing module

This module will handle functionality for writing custom rules and editing existing rules.

Automatic rule checking module

This module will be responsible for verifying rules to ensure that there are no malformed or misconfigured rules that may potentially break or cause a halt on a Snort sensor.

3.1.2 High-level Use Cases

Since the program workflow is heavily centered around the database module, and the major parts of the system revolves around the importing of rules and then distributing them, we began developing our use cases around those major movements. The most central use cases for this system are listed below. Additional use cases are available in Appendix E.

Use case	**Manual rule update**
Primary actor	User
Purpose	Update the central rule database with the newest rules and revisions.
Description	The user commands the server to update the rules, either from a local file or from external sources. When the latter is the case, the server pulls rule files from all the sources specified in the list of sources. In both cases rules are processed and stored in the database.

Preconditions	Either a file or one or more sources must exist.
Postconditions	The central rule database is up to date with the newest rules and revisions.

Use case	**Automatic rule update**
Primary actor	Daemon
Purpose	Update the central rule database with the newest rules and revisions.
Description	At configured time-intervals, the system daemon triggers a rule database update. The server pulls rule files from all the sources specified in the list of sources to be updated at the given interval, processes the rules and then stores them in the database.
Preconditions	One or more sources must exist.
Postconditions	The central rule database is up to date with the newest rules and revisions

Use case	**Manual rule distribution**
Primary actor	User
Purpose	Distribute rules to sensors as per the settings in the database.
Description	The user sends a command to distribute rules to either all or specified sensors. The server then begins the distribute routine, and eventually all sensors will be up to date with the rules and rulesets they should have.
Preconditions	Rules and sensors must exist

Postconditions	Specified sensors contain rules as per settings in the central database.

Use case	**Automatic rule distribution**
Primary actor	Daemon
Purpose	Distribute rules to sensors as per the settings in the database.
Description	Configured at intervals, the system triggers a command to distribute rules to specified sensors. The server then begins the distribute routine, and eventually all sensors will be up to date with the rules and rulesets they should have.
Preconditions	Rules and sensors must exist
Postconditions	Specified sensors contain rules as per settings in the central database.

Use case	**Enable/Disable a rule or ruleset**
Primary actor	User
Purpose	Enable or disable one or more rule or ruleset on one or more sensors.
Description	The user either enables or disables a rule or a ruleset so that the rule is either on or off on certain sensors. This status is then shared with the sensor and the sensor syncronizes as needed.
Preconditions	Rules and/or rulesets must exist.

Postconditions	A rule or ruleset has been modified to be enabled or disabled and the change is syncronized to affected sensors.

Use case	**Set rule threshold or suppression**
Primary actor	User
Purpose	Set or remove a threshold or suppression for a rule on one or more sensors.
Description	User sends a command with either thresholding/suppression parameters or removal of a threshold or suppression for a rule on one or more sensors. The system updates the database and distributes the change to the sensor(s).
Preconditions	Rules and sensor(s) must exist.
Postconditions	Rule tuning is saved in the central database and distributed to affected sensors.

Use case	**View rules**
Primary actor	User
Purpose	Display rules to the user.
Description	Rules are retrieved from the database, processed for GUI format and then delivered to the GUI.
Preconditions	Rules must exist.
Postconditions	The user is informed of the current status of the central database and the status of rules and rulesets enabled/disabled on sensors.

3.1.3 Detailed Use Cases

Use case	**Central rule-database update**
Primary actor	Server
Purpose	Update the central rule database with the newest rules and revisions.
Description	An update can be triggered either by the user or the daemon. The system retrieves the files in question, checks what is new and what is not and then updates the database accordingly.
Preconditions	Depending on the request, this process requires a local file or the server must be configured with external source(s).
Postconditions	The central rule-database is up to date with the latest rules and revisions.
Triggers	Manual rule update, Automatic rule update, Write rule

Basic Flow:

1. Update is triggered.
2. Rule Downloading Module compares hash values from external sources with last update.
3. Updated rule files are downloaded from sources. The files are validated and passed to the Pre-Storage Processing Module.
4. The new files are parsed for rule SID and revision number.
5. A list of rule SID and revision numbers are gathered from the database and these are compared to identify which rules are new or modified.
6. New or changed rules are then parsed into objects and these objects are passed to the Database/Core Module.
7. The Database/Core Module stores the new or changed rule objects into the database. The user is notified that the update is complete and given a list of changes.

Extensions:

1. (a) Update is triggered by user.
   (b) Update is triggered by daemon.
2. (a) The hash-values from all sources are identical with previously stored values. Database is up to date and the update process terminates. User is notified that database is up to date.
   (b) The hash value from one or more sources is newer than previously stored values. Database must be updated, and the user is notified that the database is not up to date and an update process has begun.

Error handling:

1. The source and database checksum is different but the source rules are older than what is stored in the database: *The program can still continue, because if the SIDs already exist they will not be added again and if the revisions are older the rules will also not be updated.*
2. A ruleset has changed its name or one or more rules are moved to a different ruleset: *Based on a configuration file, the affected rules will either be a) automatically moved to the new ruleset or b) stay put in their original ruleset. The user will be notified and can choose to reverse or distribute the changes in either case.*

Use case	**Distribute rules**
Primary actor	Server
Purpose	Distribute rules from the central database to the sensors.
Description	After a command to synchronise is sent either by the user or the daemon, the server calculates and organizes the central database data and synchronises this information with the affected sensors.
Preconditions	One or more rulesets or rules must be present in the central database. One or more sensors must be registered.
Postconditions	The rules on the sensor(s) are synchronized to reflect those in the central database that are enabled/disabled for those sensors.

Triggers	Manual rule distribution, Automatic rule distribution

Basic flow

1. Synchronise signal is sent.
2. The system calculates and organizes the data for each sensor affected by the synchronisation.
3. The system initializes a synchronisation protocol with the affected sensor or sensors, sending a list of what the sensor should have.
4. The sensor calculates diff between should have and has.
5. Compare diff.
6. Send update.
7. The sensor includes the new changes into its database and files and tells the central system that it is now up to date.
8. The system carries out maintenance to reflect the synchronisation.
9. The user is notified that all sensors affected are now up to date.

Extensions

1. (a) User sends a command to synchronise one, more or all sensors
   (b) The system reaches an interval for synchronising all sensors.

   enumi4
2. (a) The sensor content is different from what it should have, asks central system to send what it needs.
   (b) The sensor content is not different from what it should have, no further action is required. This status is sent to the central system.
3. (a) The system retrieves and sends what the sensor asks for.
   (b) The system terminates the synchronisation protocol with the sensor in question.

Error handling

1. The change may never be synced to the sensor due to errors: *There must be an error protocol that is followed in this scenario.*

3.1.4 Use Case Diagram

The use case diagram is presented in Figure 4.

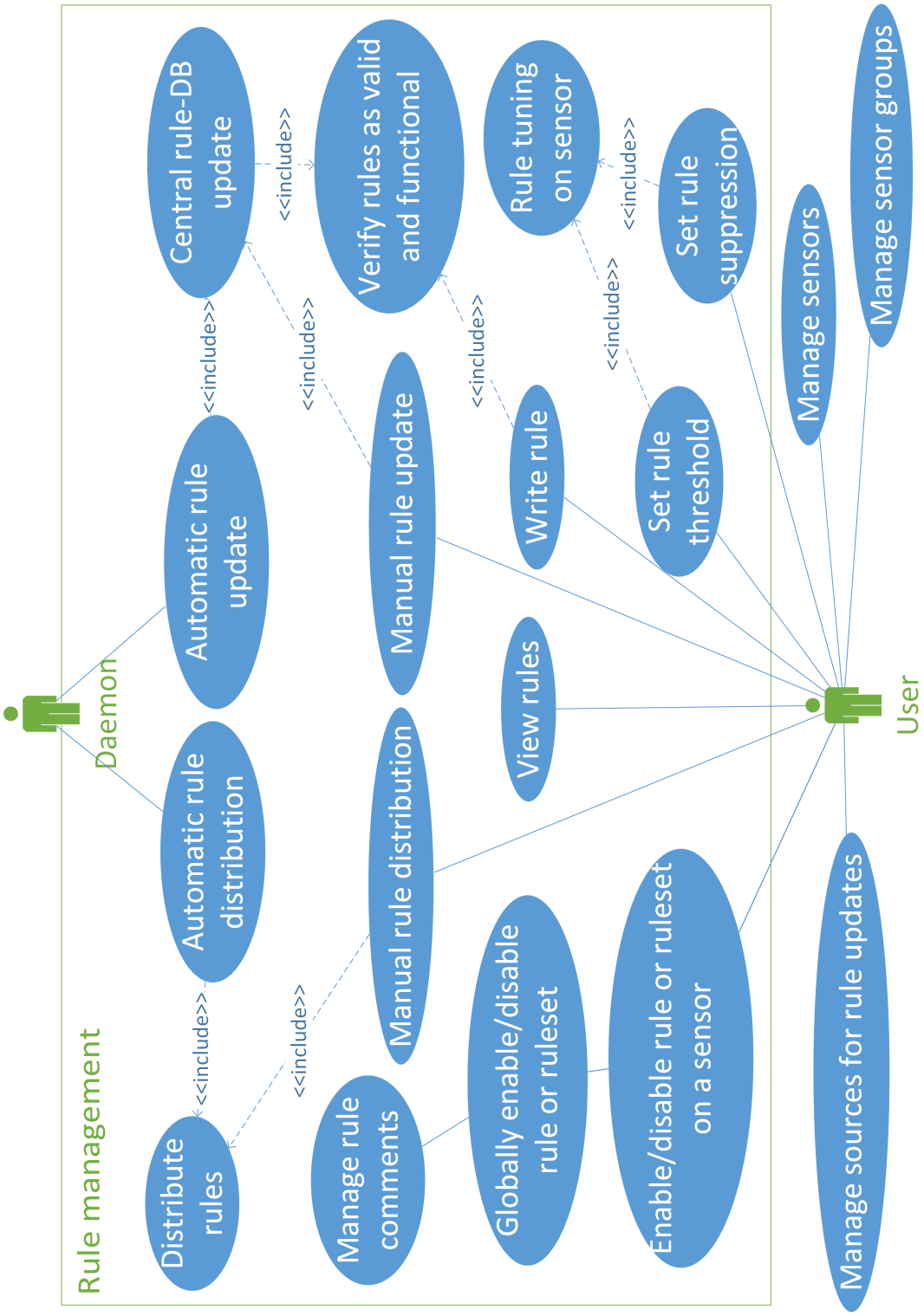


Figure 4: Use case diagram

3.1.5 Domain Model

After outlining all the concepts above, we created the conceptual class diagram in Figure 5, which shows the classes needed in the system. Even though a Snort rule is quite simple by itself, there is a lot of meta-information surrounding it, such as rule revisions and rule classifications. Therefore, the system needed at least 16 different classes in order to fullfil the basics of the required functionality.

3.2 Supplemental Requirements

In addition to the features discovered in the use cases, we added some requirements we felt were needed, that can be broken down into a RUP FURPS+ fashion [24].

3.2.1 Functionality

- The system should be able to restart a Snort sensor in the event that one crashes.
- The system should be able to detect if a Snort sensor is not responsive over the network.
- The system must be portable to the majority of the popular Linux distributions. Porting to other operating systems, such as Windows, should be possible through changing the operating system-specific code.

3.2.2 Usability

- The graphical user interface must be simple and clean, but still contain all the features any user might require to interact with the back-end system.
- The system interfaces, herein API and function calls, must be self-explanatory.

3.2.3 Reliability

- The system must not cause a Snort sensor to crash.
- The system must be responsive to API-calls at all times. In the event the system capacity limit is reached, new requests should be queued.

3.2.4 Performance

- The system should handle administration of at least 200 sensors.
- The database should be able to maintain at least 100 000 individual rules.
- Any operation in the system in normal operations (i.e. after the first big rule update) should take no longer than 1 second on average.
- The relationship between time and workload should be sublinear, meaning that if a single operation takes 1ms, ten operations should take 10ms or less.

3.2.5 Security

- The link between the system server and the remote sensor must be secured in such a way that a man in the middle cannot read the transmission in plaintext.
- The remote sensor and the system server must verify that they are trusted parties in the system, through authentication.

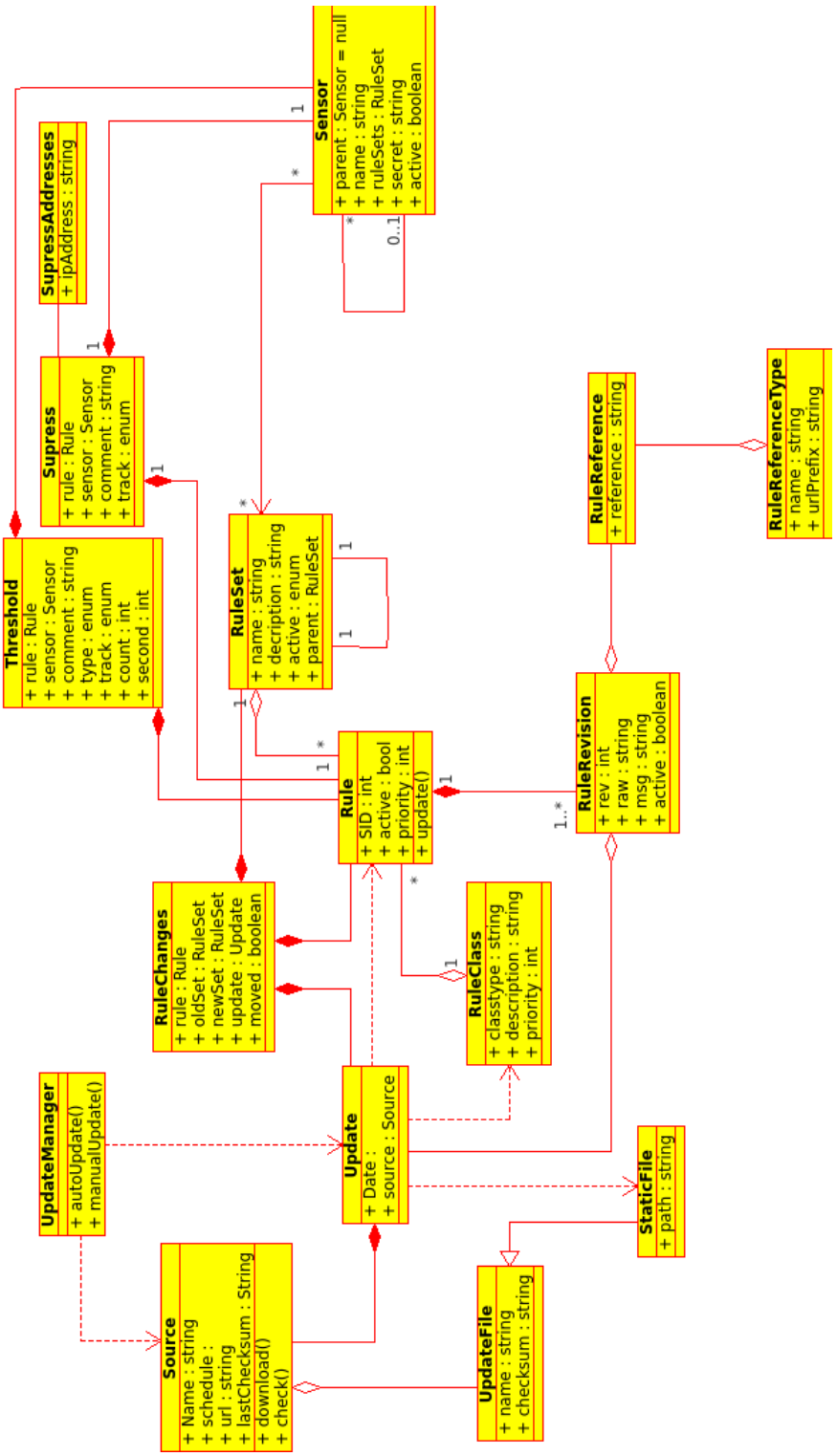


Figure 5: Conceptual class diagram

- Certain important API-calls must be protected against misuse from a non-trusted party.
- All processes and files related to the system must exhibit the principle of least privilege [25].

3.2.6 Interoperability

The system must be able to work with Snort in such a way that it can import new files to the Snort configuration files structure and must be able to reset Snort in such a way that the new configurations are loaded.

3.2.7 Licensing

The system will be released to the public as Open Source under the GNU General Public License v3 license[26].

3.3 Constraints

3.3.1 Platform

As mentioned in the project plan (Appendix A), Snort is primarily used in conjunction with the Linux/UNIX operating systems. For that reason, the project was restricted to only develop for that platform. More specifically, we limited ourselves to the Ubuntu and Debian operating systems, which are the primary platforms Avdeling BKI uses as well. We have, however, strived to make the software portable to most Linux distributions. A port to Windows should also be possible through changing some of the operating system-specific code.

3.3.2 Data Security

While the system itself will not handle any data that may contain personal data, it may contain rules and rulesets that are considered classified or confidential. The system has therefore been built with this in mind and ensures confidentiality within the boundaries of the system. The system does not guarantee the confidentiality of the data outside these boundaries. The boundaries are defined as the program and its processes and threads, and the network link established between the system and remote sensors. The system assumes that the users have already established database security.

3.3.3 Graphical User Interface

The project has had its main focus on the system functionality and efficiency, and developing a GUI has been considered a secondary task and was prioritized as such. None the less, a GUI for demonstrational purposes has been included.

3.4 Security Assessment

During the inital phase of the project, we conducted a small security assessment in order to be aware of and possibly fix any security issues the software might have. We created a simple risk analysis table of the potential weaknesses we could find, listed in Table 11.

Threat	Component	Consequence	Probability	Risk	Strategy
Unauthorized users change data through the API/web-interface	API/web	8	5	6,5	Authentication/authorization. Django support for users and authentication.
An unauthorized user gains direct access to the database and is able to alter the data.	Central database	10	1	5,5	Assume database security is in place.
Continually sending requests to the server, filling it up with requests to the point of causing a denial of service.	RPC Server	7	4	5,5	Authenticate requester, Rate-limiting on requests, logging.
An attacker inherits escalated privileges through exploiting the software if it is run as root or other high privilege user.	All processes	8	3	5,5	Follow least privilege principle. Python also protects against buffer overflow
The SID/Rev of a malicious rule is newer than an existing rule, with the purpose of overwriting the existing rule.	Update module	7	2	4,5	Add possibility for the user to see changes before they are commited.
In a scenario where the database is not on the localhost, a man-in-the-middle attack is possible.	Central database	6	3	4,5	Assume database security is in place, encrypt transmissions.
Continually sending requests to the sensor client to update itself, filling it up with requests to the point of causing a denial of service.	RPC Client	5	4	4,5	Limit the allowed update frequency, logging.
Some unauthorized machine poses as a legitimate sensor and gains access to rules through update requests.	RPC Server	5	2	3,5	Authenticate the sensors.
During an update, one or more rules contains errors that causes Snort running on a sensor to crash or shut down.	Update module	8	4	6	Verify the rules.
A rule file is specially crafted to contain an SQL injection.	Update module	10	2	6	Use prepared statements and never let input directly touch the database.
SQL injections, XSS og CSRF (cross site request forgery)	API/Web	5	5	5	Django has support for protection against these attacks. Validate input.

Table 11: Security Assessment

4 CONCEPTS OF SNORT

This chapter gives a brief introduction to the basic concepts of Snort.

4.1 Rules

A Snort rule is made up of a header section followed by rule options. The header specifies which network, protocols and ports the rule is valid for, and which action to take if the rule matches traffic, while rule options provide metadata for the rule, such as its ID and classification [19]. The rule options are specified as plain text in the *rule string* enclosed in parentheses (Listing 4.1). Note that a badly formatted rule string given to Snort will cause a crash.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 69 \
(msg:"PROTOCOL-TFTP Get"; classtype:bad-unknown; \
sid:1444; rev:9;)
```

Listing 4.1: Example rule string.

There are several types of rule options, but only the general rule options are relevant for this project. The general rule options provide information about the rule, such as signature ID, revision number and classtype. Other types of options, such as payload options, which are actively used in detection, are not relevant for the management of rules and are thus not considered in this project. Some of the general options can only be specified within the rule string, referred to as *inline* specification, while others can be specified in separate files, referred to as *external* specification.

A rule can either be enabled or disabled in the detection system, meaning that the rule will only have an effect on detection when it is enabled. Since this project is dealing with multiple sensors connected to one central database, this functionality has been expanded so that users can keep a rule enabled on one sensor and disabled on another. In addition, the user can also set an enabled/disabled flag on *rulesets*.

4.2 Rulesets

The term ruleset, sometimes rule set, is not formally defined in the Snort Users Manual, however it is used a few times to refer to the complete collection of official Snort rules[1] and to the complete set of rules applied on a Snort system. In practice, rules are categorized by intrusion type (e.g. malware, trojan and worm) in order to keep the rules structured. These groups of rules are often named by their intrusion type, and are also referred to as rulesets. Snowman and this report use only the latter definition of rulesets. The official Snort rules are split into files whose names denote the ruleset.

4.3 Generators

Generator is a term used for any subsystem of Snort which generates alerts from rules [19]. Each generator is identified by a unique generator ID. The rules subsystem, which has generator id 1, is the only relevant generator for this project.

[1] Also referred to as the Sourcefire VRT Certified Rules, see http://www.snort.org/snort-rules

4.4 References

Snort contains a plugin which allows rules to contain references to external attack identification systems, which provide additional information about the alerts [19]. A reference is basically a URL string pointing to a webpage containing this information.

4.5 Event and Detection Filters

Filtering can be applied to rules in order to control the amount and frequency of alerts. Two types of filters can be applied to a rule: event filters and detection filters. A detection filter can only appear inline, whereas an event filter can only be specified externally[2]. All filters operate with a *count* and *seconds* variable, representing the number of events in a time-interval and the duration of the time-interval, respectively. Event filters are further divided into three types (table 12). Detection filters are filters limiting the number of times a rule generates an event, while event filters limit how many events are producing alarms.

4.6 Rule Suppression

Some rules might match traffic that is unwanted from most hosts, while some hosts would be allowed to send traffic matching the same rules. In this case, Snort gives the possibility to suppress alarms when the source or destination of the traffic matches the configured parameters.

Filter type	Alerts
Event filter	
Limit	One for every event when number of events $\leq$ C in interval S
Threshold	One for every C events in interval S
Both	One per interval S when number of events $=$ C
Detection filter	One for every event when *count* $>$ C in interval S

Note: C and S represent the count *and* seconds *values in the filter.*

Table 12: Effects of different filter types.

[2]An event filter can currently be written inline (called a rule threshold), but according to the Snort Users Manual this feature is deprecated and will not be supported in future releases.

5 DESIGN

This chapter presents the design specification of Snowman.

5.1 General Design

This section is dedicated to the discussion of the general design of Snowman.

5.1.1 Centralized or Distributed Setup?

We decided early on to design Snowman to be a distributed system. The main considerations behind this choice were the bandwidth usage needed by the rule updates, and the load on the central server. Figure 6 shows the difference between a centralized and a distributed setup. The red dotted lines represent the communication in a centralized setup, while the blue lines between the server and the clients represent the communication in a distributed setup. The latter is the principle used in Snowman.

Bandwidth Requirements

The Snort sensors managed by Snowman might have varying network infrastructure connecting them with the central server. In a worst case scenario, the sensor might be located at a military base in the field, on the other side of the planet, only connected to the central server via a satellite connection with limited capabilities for transmitting data. As the rulesets applied to sensors typically consist of tens of thousands of rules and the changes between two versions of the sets typically only affect about a thousand of them, it seems reasonable to only copy the changes when an update should be done. This is achieved by using a distributed system.

Central Server Load

When a full update of all sensors is requested, new rulesets would have to be created and distributed to the sensors. In a centralized environment, the server would need to generate all the new rulesets and send them to the sensor. The server would not know beforehand which rules currently exist on the sensor, so a full list of rules would need to be sent, even if there were no changes to any of the rules. Also, if the sensor for some reason is unavailable, or connected through an unreliable network connection, the central server would waste a lot of extra effort to send the rules to the sensor, even if it is not neccesary as no rules have changed.

In a distributed environment, we would be able to put some logic (a client) into the machines hosting the sensors. This way, when an update is occurs, the client can get a list of rules from the central server, and on its own accord be responsible for collecting the new revision of the rules that have changed or are missing. This way, the central server would only need to transmit the rules from its database if the rule is actually needed by the sensor.

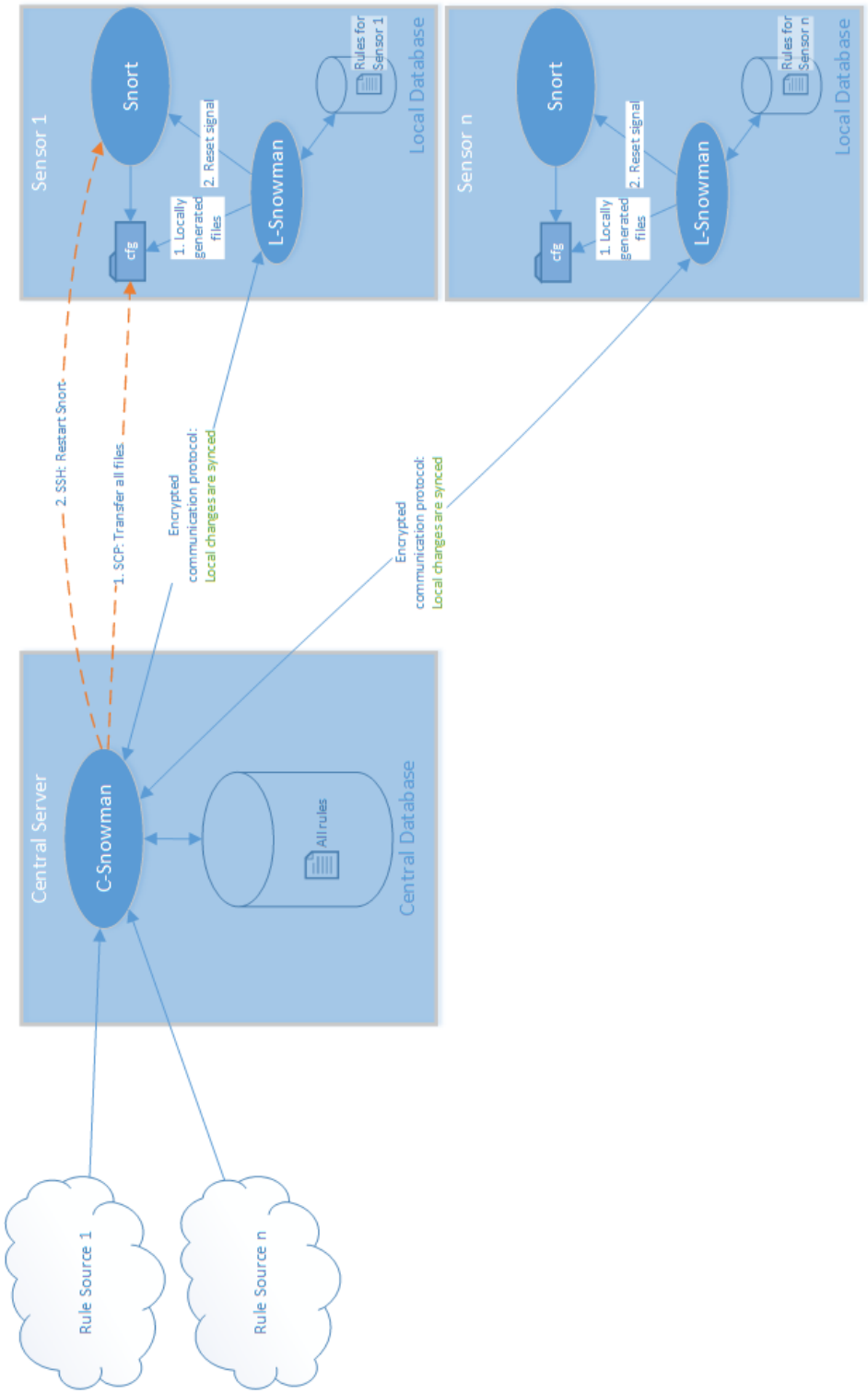


Figure 6: General architecture

5.2 Server Design

The server is the central component of Snowman and generally consists of:

- Central rule storage
- Web interface for user interaction
- Update module to get updates from rule sources
- RPC interface to communicate with the clients

The central storage of rules would be in a normal relational database. As we have decided to use Django (described in Chapter 6) as our framework, we have a great flexibility in selecting the database we actually use. The current version of Django (1.6) supports MySQL, PostgreSQL, SQLite and Oracle. For the RPC interface, we have decided to use XML-RPC. This is a very simple protocol using XML over HTTP to transfer requests and data to and from the server. To be able to keep up with the aspect of privacy and security, we encrypt the raw HTTP stream using SSL.

5.2.1 Architecture

We have designed the server by the layered approach of MVC (Figure 7). The web interface uses pure Django-logic. The HTML visible in the web browser is generated by the Django template-engine, which recieves its data from the views created by using ordinary Python code. The XML-RPC server uses a handler which administers the connections to and from the sensors, and exposes the relevant methods to allow the sensors to perform synchronisation. The update module is used for unpacking and preparing the update files for parsing. It then handles the files to the parser. This parser is then tasked with translating the raw rule files into more managable data structures for the Updater, which in turn saves them to the database. Finally, common for all the interfaces are the Django Models. We use the database abstraction layer provided by Django to map the rows of a relational database into Django objects. This is the part that lets us write code, which is compatible with several different database engines.

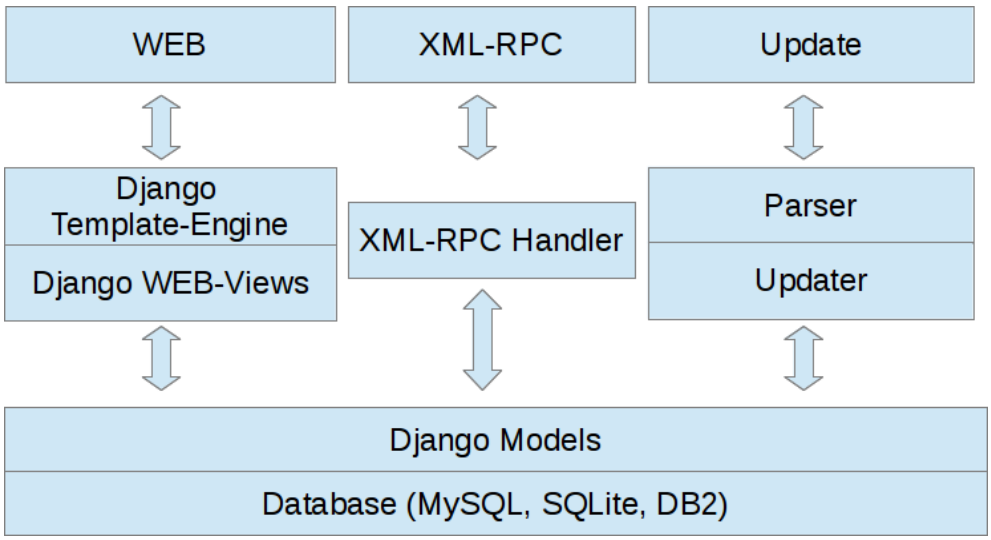


Figure 7: Server Architecture

5.2.2 User Interface

The main user interface for Snowman is the web interface. It is based on Django's template system, which generates the HTML that is displayed in the web browser on the user's computer.

URLs

The first step of serving a page to a user is an HTTP-request for a specific page. The pages are uniquely identified by URLs, such as '<server address>/web/rules/'. The URLs are structured in a logical manner, where everything in the web interface is found under /web/, everything regarding the administration of rules is found under /web/rules/ and so on. The URLs might contain dynamic data, such as page numbers (e.g. /web/rules/rules/page/3/ which will give you the third page of the rule-list), so that dynamic content can be loaded when needed. Django uses the URLs to identify which view the requested url corresponds to. When a view is identified the request is passed to the corresponding view.

Views

The views constitute the part of the user interface that collect the required data from the models-layer and then generate a response to the web client. The response can contain a full web page, or just a small subset of one. The typical steps a view makes to generate a response would be:

1. Collect the required data from the database(s) using the Django models.
2. Deliver the collected data, and a template, to the template-engine to generate the HTML representing the webpage the user requests.
3. Send the created web page as a response to the user.

The view system decides what to show and then lets the template system decide how to show it. The view system by itself does not create any HTML or similar; It solely creates data structures. If the request was for raw data and not a webpage, step 2 in the list above might be skipped, and instead return the data as some form of recognizable data structure in step 3. If errors occur in the view system, appropriate exceptions can be raised that tell Django to return an error code or an error page to the user instead of what the user requested.

Templates

When the view system has prepared all the data, the template system takes over. The task of the templates is to take the supplied data and inject it into pre-made skeletons of HTML. This way, all the design/display parts of the web interface are contained into one single and separate place, and that gives us a very clear line between data and design.

5.2.3 Database

As the database of the server is quite complex, this section contains one figure per module.

Core

The most central part of our system is the representation of a **`Rule`**. A single **`Rule`** might have several `Revisions`, as we might want to keep some history when they get updated. Each `Revision` has its own set of `References` attached. A single **`Rule`** also has a lot of

other properties which it might share with other Rules. A Generator, a Classification and a Ruleset are examples of such. Finally, the Rules are assigned to a Sensor by attaching Rulesets to the Sensor. The core database architecture used for storing the rules is described in Figure 8.

Tuning

In addition to storing the raw rules, we would like to be able to store tuning parameters on the rules. As sensors might have different tuning parameters, we cannot attach the tuning directly to the rules. Separate tables (as described in Figure 9) are thus needed to model the tuning parameters. Both types of filters and the suppressions are connected to a Rule and a Sensor. This way, we let the user specify filters for the individual sensors. If a sensor does not have a filter applied, but any parent sensor of that sensor has one, the closest parent's filter will be applied.

Update

When rules are inserted into the database, we would like to have some traceability to be able to determine where and when a rule has entered our system. To be able to do this, we have the update object (Figure 10). The structure of the update module allows us to store the update sources, and adds a new Update object to each update executed from a given source. These Update objects can in turn have relations to Rules, RuleSets and RuleRevisions, so that we can track changes for each update.

5.3 Client

As the client is just going to take care of getting rules for a single sensor from the central server and should just apply these to one sensor, the architecture of the client can be much simpler than that of the server.

5.3.1 Architecture

The MVC architecture for the client can be seen in Figure 11. The update handler of the client has both an XML-RPC Client and an XML-RPC Server. The XML-RPC Client is used to connect to the central server to do a rule synchronisation. The XML-RPC Server is here so that the central server can connect to the client and check to see if the client is active or to order the client to perform rule syncronisation.

As the client should be as lightweight as possible and the only aspect we need from any framework is a simple database connection and abstraction layer, we have decided to not use Django for the client. Instead we use a database ORM [27] for Python called SQLAlchemy [28], which is much more lightweight than Django for this purpose. The client also contains a file creator which takes the rules objects in the database and generates the files Snort needs in the format Snort needs them in.

5.3.2 Database

The database on the client can be significantly simpler than the one on the server, as we only care about the rules which should be present on a given sensor. The full database schema is available in Figure 12. The central part of the data structure is the rule table. This table contains all the parameters directly related to the rule. In addition, we have separate tables for all the meta parameters surrounding the rule, such as rulesets or filters.

Figure 8: Core server-database

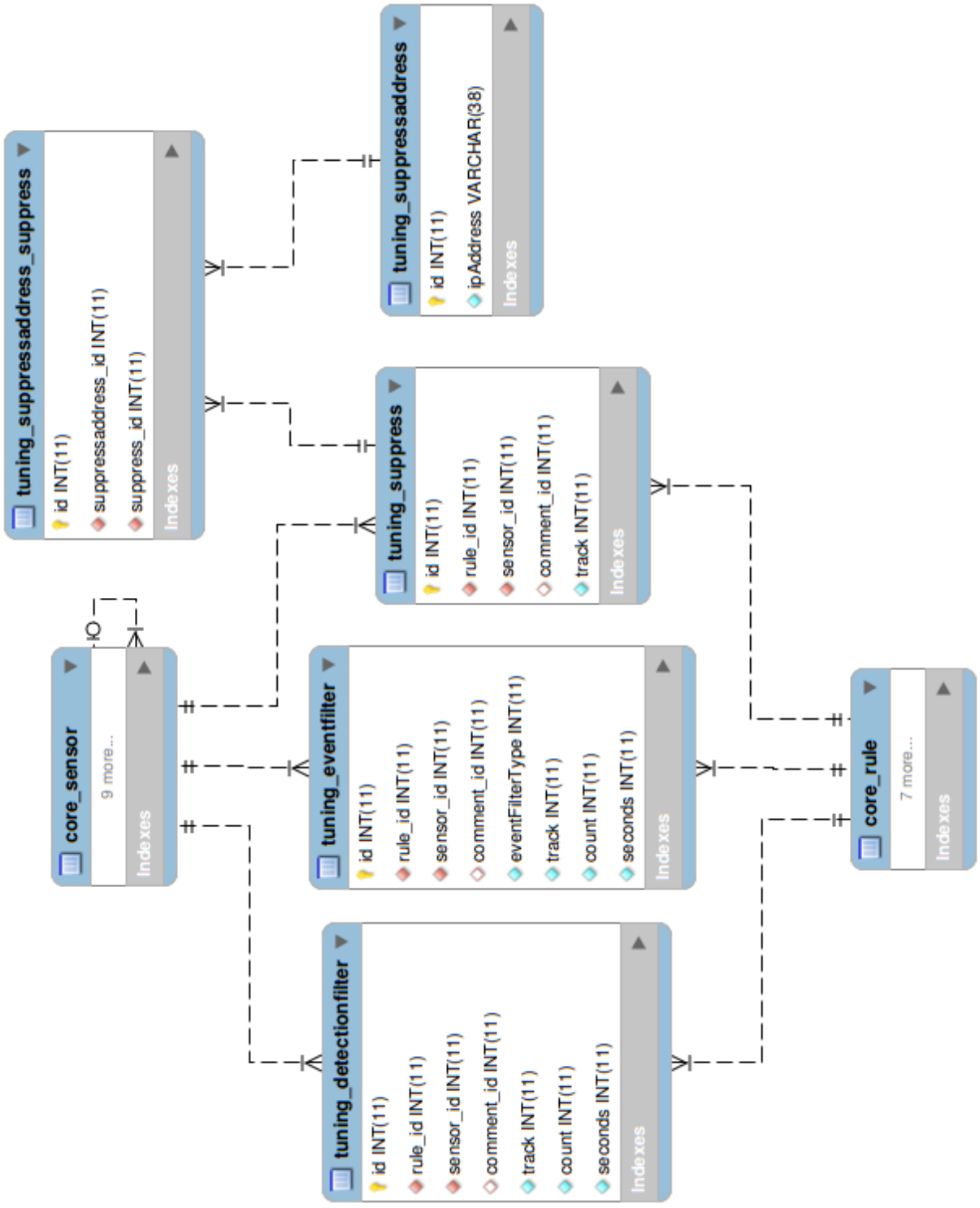


Figure 9: Tuning server-database

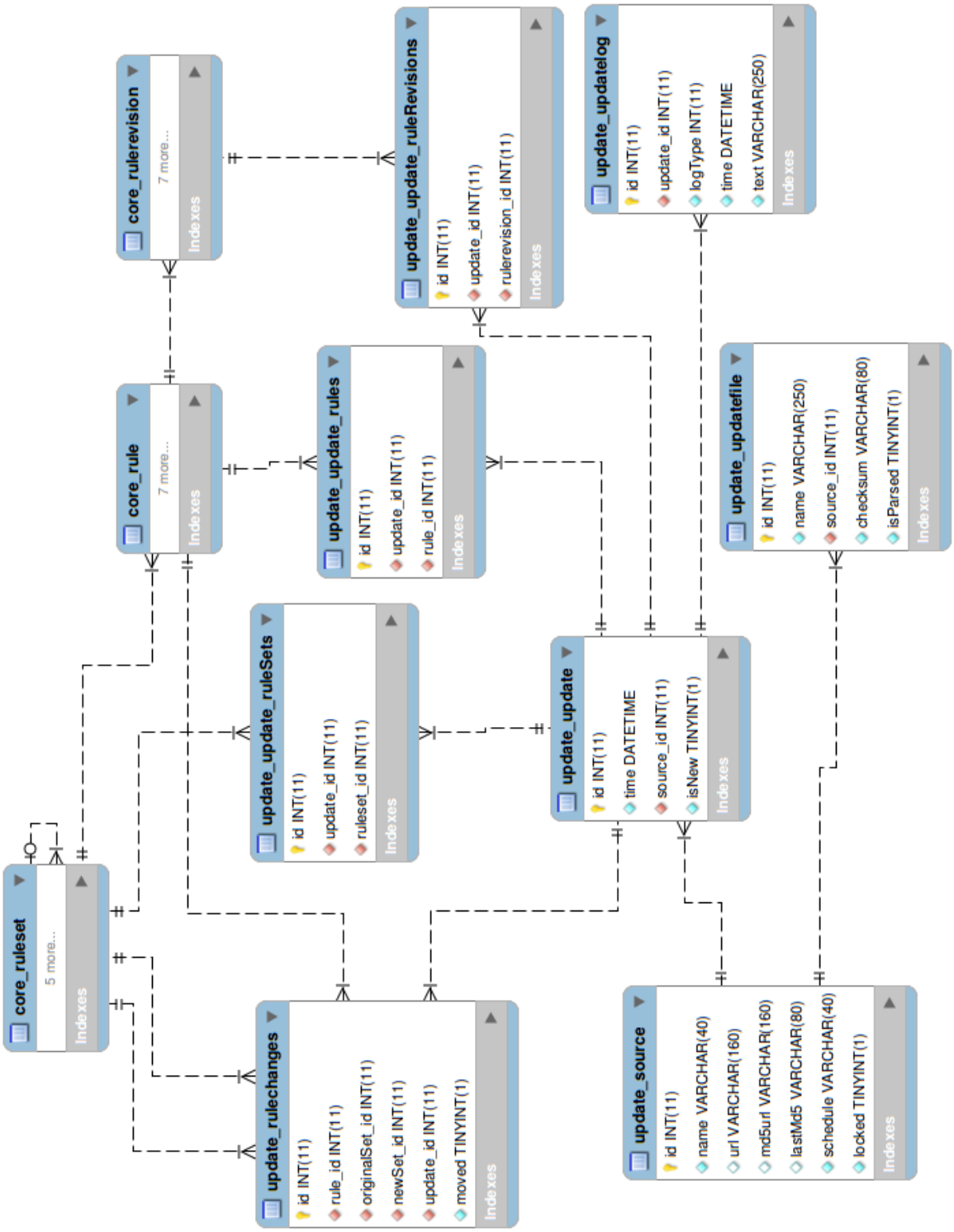


Figure 10: Update server-database

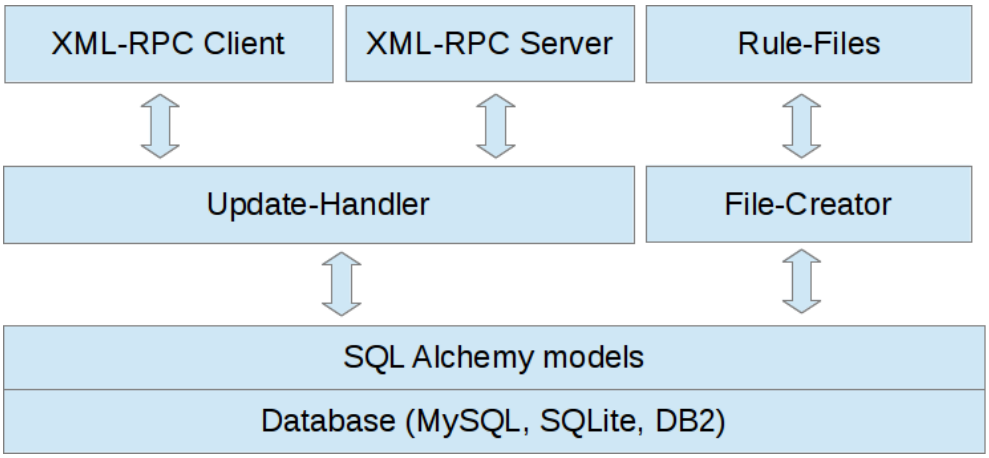


Figure 11: Client Architecture

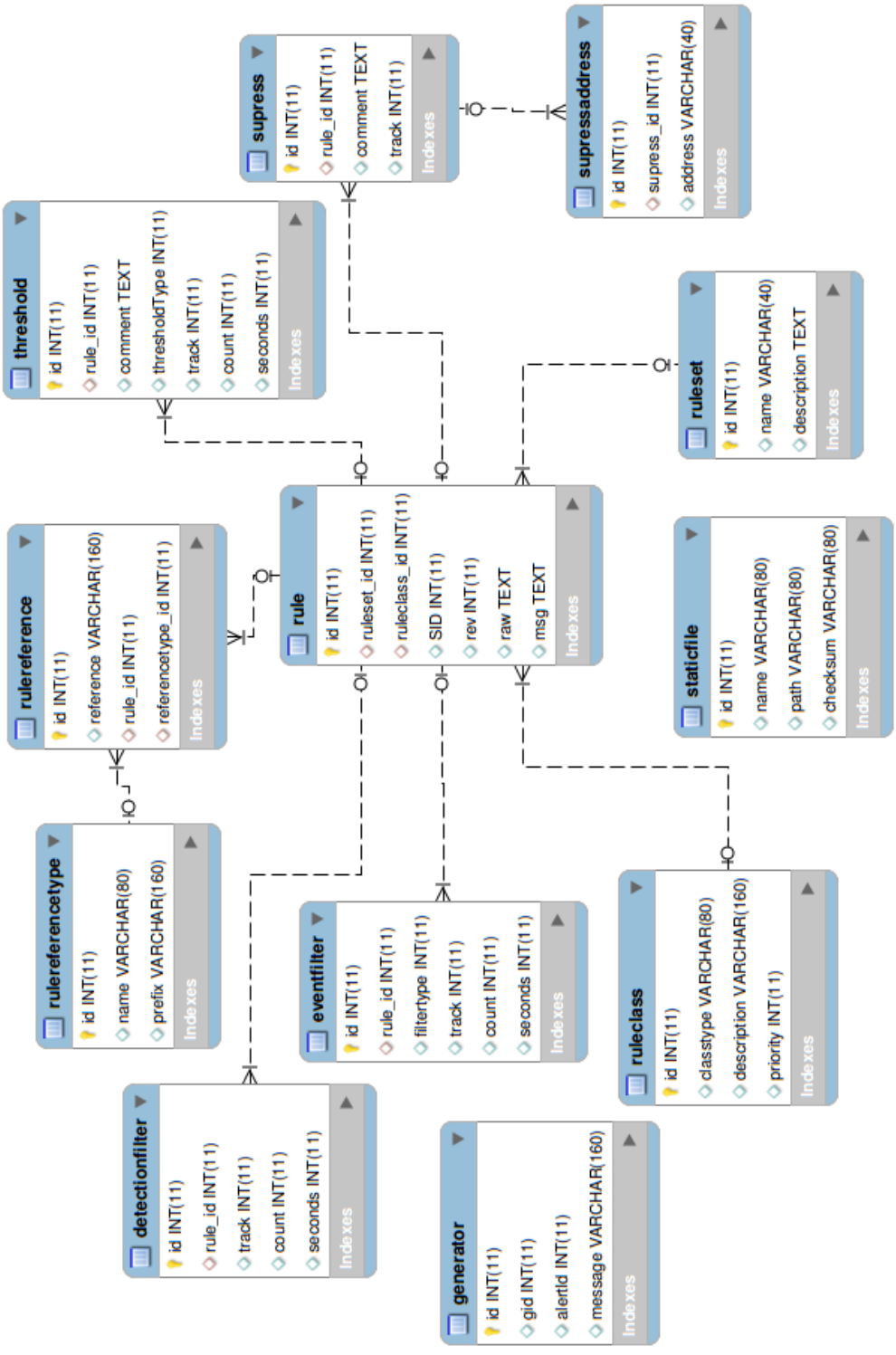


Figure 12: Client Database

6 IMPLEMENTATION

This chapter focuses on the technical implementation of Snowman. The first sections cover programming languages, development tools and frameworks used in this project, followed by a description of the most important modules of the server, including the GUI. Application logging and deployment of the software is covered at the end of the chapter. Some sections within this chapter require general knowledge of the key concepts of Snort described in Chapter 4.

6.1 Programming Languages

Snowman is primarily written in Python 2.7 [29]. The project group landed on this decision after discussing the aptness of C, C++, Java and Python to this project, with regard to development and processing efficiency. Due to the limited time-frame of this project, it was vital to get right into the development without having to learn a new programming language or unfamiliar programming concepts. It was also important to find a language which produces high performing programs, since this project aims to be more effective than similar software. The C and C++ languages were considered because these languages can produce programs with very high performance, however, these were abandoned as we concluded that they would induce more work because of their complexity, and thereby be too time-consuming.

The final decision fell on Python mainly because it lets the programmer develop complex programs in a short amount of time. Python has automatic memory management and offers a wide range of built-in tools, which gives the developers time to focus more on the code and functionality rather than overhead tasks like managing memory and including external libraries. A concern was raised that Python, being a high-level language, might not be efficient enough for this project. This turned out to not be a problem, as Python can run modules written in C. A major reason for using Python was also Django (see Section 6.3), which is a Python framework used in this project that greatly simplifies database transactions and generation of web pages.

The most current version of Python available at the start of this project was version 3.3. The project group selected version 2.7, as a disparity in compatibility [30] between 2.X and 3.X versions of Python cause some problems. According to a recent survey [31] Python 2.X still holds a dominant position among developers, five years after the release of Python 3.0. According to [30], version 3.3 has slightly worse library support. The continuing popularity of 2.X versions implies that many packages are still being developed and maintained for these versions, and these versions only. One example of this is the aforementioned Django framework which is heavily used by Snowman. Development using Python 3.X with Django is possible with the use of a compatibility layer [32], but the project group considered this as a bit cumbersome. Taking into account that the majority of the group was already comfortable with Python 2.X, this came out as the most time-saving choice. The biggest drawback of this decision is that it does not make Snowman future-proof, and it will probably have to be ported to Python 3.X in a long-term perspective.

For Web-programming, this project has used HTML and CSS for the design and JavaScript for client-side scripting.

6.1.1 Standards and Guidelines

Python imposes a strict indentation regime, as this is actively used in the syntax to delimit blocks of code. The project group has followed standard Python guidelines for best practice programming, with principles such as DRY[1] and EAFP[2] (the latter actually caused some unforeseen performance issues described in Section 6.5.6).

Documentation has been a high priority. The project group has aimed to make the code self-documenting by utilizing sensible names on variables, functions and classes to enhance the reader's understanding. As a general rule, variable names are written in camelcase[3]. With a high focus on development, the project group has taken advantage of Python Docstrings [33] (documentation strings) which are incorporated into the code, and can therefore be written at the time of coding without taking the focus away from the programming. The Docstrings describe how the code is used, and form the basis of Snowman's documentation. Listing 6.1 demonstrates the EAFP-principle, also note the use of a Docstring at the beginning of the method.

The Python manual encourages the use of exceptions, which is a useful way of sending messages up the execution stack when an error or irregular behaviour is detected. Customized exeptions are raised in Snowman for example when parsed data has a wrong format (*BadFormatError*), when critical objects are missing from the database (*MissingObjectError*) or when an unsupported rule is encountered (*AbnormalRuleError*).

Writing object-oriented code has not been a goal in this project, but concepts such as encapsulation, polymorphism and inheritance have been applied when needed.

```
1  def getSecondMessage(messages):
2    """Returns the second message from messages or
3    empty string if second message does not exist."""
4
5    # Below code demonstrates the principle of
6    # EAFP (1) versus LBYL (2); The execution
7    # of block 1 is faster than block 2 when
8    # message[1] exists. Block 1 is therefore
9    # preferred whenever the try-block is
10   # expected to succeed without an exception.
11
12   # Block 1, EAFP:
13   try:
14     message = messages[1]
15   except IndexError:
16     message = ''
17
18   # Block 2, LBYL (look before you leap):
19   if len(messages) >= 2:
20     message = messages[1]
21   else:
22     message = ''
```

[1] "Don't repeat yourself". The principle that code should not be duplicated in a project.

[2] "Easier to ask for forgiveness than permission".

[3] A style convention where multiple words are written as one, with the first letter of each word capitalized, e.g. getAllRules.

```
23
24    return message
```

Listing 6.1: The EAFP-principle.

6.2 Development Environments

This section describes the various development environments and tools used in this project.

6.2.1 Environments

Snowman is developed on and for the Linux platform. Eclipse [34] was selected as the default development environment, as this is freely available and contains the major features required for effective development such as project management, a Python editor and a debugger (available with the PyDev add-on). PyUnit has also been used for unit testing. In addition, the TeXlipse and PDF4Eclipse add-ons for Eclipse have been used to write and generate this report with Latex [35]. Since Python is an interpreted language, the Python interpreter has been used throughout the project to test and debug code segments. The user interface has been mostly programmed for and tested in the Chrome web browser as it has a very good built in debugging environment, but limited testing has also been conducted on Firefox and Internet Explorer.

6.2.2 Version Control

This project has used Subversion (SVN) provided by the IT department at Gjøvik University College to keep version control on all project files. This includes all source files and project documents such as design documents, meeting minutes and the report. The SVN repository hosted by the IT department has also been used to ensure that data is not lost.

6.3 Frameworks

To save the project time and effort programming generic tasks, we decided to utilize several frameworks to accomplish our goals, as outined in this section.

6.3.1 Django

To facilitate database transactions and the creation of the Web interface, this project uses the Django [17] framework. Django is an extensive web-framework, designed to make the process of building web-based applications and their backends much more efficient and simple. It takes care of the database ORM, which controls all the database-transactions, and offers us a simple and clean abstraction layer. Django is also providing a template-system for the web front end, which make a clear dividing line between the data and design of the webpage.

6.3.2 SQLAlchemy

A database ORM would also be needed on the Snowman-client, but the rest of django is not needed there. Because of that, SQLAlchemy is used on the client.

The SQLAlchemy framework creates an abstraction layer on top of the database just like Django. It also lets us create the code without worry of database specific query code, so that the user is free to pick and choose which database they want the system to operate with.

6.3.3 jQuery

When it comes to using JavaScript for client-side scripting, jQuery [36] is a good solution, as it provides a framework for common JavaScript operations on the DOM[4] and contains abstractions for a lot of browser-spesific code, which gives programmers a nice and clean interface. It also makes AJAX-calls[5] a lot easier, requiring much less code compared to regular JavaScript (example in Listing 6.2).

```
1  // AJAX call in jQuery:
2  \$.get('/web/rules/');
3
4  // AJAX call in JavaScript:
5  function loadXMLDoc() {
6      var xmlhttp;
7
8      if (window.XMLHttpRequest) {
9          // code for IE7+, Firefox, Chrome, Opera, Safari
10         xmlhttp = new XMLHttpRequest();
11     } else {
12         // code for IE6, IE5
13         xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
14     }
15
16     xmlhttp.onreadystatechange = function() {
17         if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
18             document.getElementById("myDiv").innerHTML = xmlhttp
                   .responseText;
19         }
20     }
21
22     xmlhttp.open("GET", "/web/rules/", true);
23     xmlhttp.send();
24 }
```

Listing 6.2: AJAX call in jQuery and JavaScript.

6.3.4 Bootstrap

To facilitate the design and appearance of the web interface, this project uses Bootstrap [37], which is a front-end framework derived from Twitter. It provides a lot of ready made CSS and design elements that make a web page responsive and typographically readable. Consider the example job of creating a <**table**> element and spending hours setting up and tweaking the CSS for the said table element. With Bootstrap, we can just add <**table class**=‘**table table**−bordered‘> and get the table formatted nicely with colors and fonts that are typographically matching.

6.4 Core Module

This section focuses on the classes and methods in the `core` module. This module contains the most vital classes of the Snowman server including **`Rule`** and **`RuleRevision`** which hold the data of each rule in the system. Simple classes in this module such as **`Generator`**, **`RuleReference`** and **`RuleClass`**, which are connected to the **`Rule`** class are

[4] The webpage content or elements.
[5] AJAX is web development technique that allows you to send asynchronous requests to a web server.

only briefly explained in this section. For an overview of all objects in this module, refer to the class diagram in Section 5.2.3.

6.4.1 Rules and Rule Revisions

Rules in Snowman are split into two parts; the rule and the rule revision. As threats change, or errors in the rules are discovered, a rule may become obsolete and a new revision of the rule is released. The rule string, message and detection filter might change from one revision to the other, and are stored in the rule revision object along with the revision number. A revision is always connected to a rule object, which contains data that is constant for all its revisions, such as signature ID and rule set. The reason behind keeping rules and revisions in two separate objects is to avoid writing the same constant rule data for all revisions. Users of Snowman have the possibility to control how many revisions to store per rule, and whether incoming revisions should be active or inactive by default. This feature allows users to control automatically downloaded rules before they are put into the detection system. Users can also manually activate or deactivate rule revisions, and Snowman will distribute only the newest active revision of a rule.

6.4.2 Rulesets

As stated in Chapter 4, the categorization of rules in rulesets is a de facto way of organizing rules. The community rules[6] contain rules with a "ruleset" option within the "metadata" option in the rule string, which specifies the ruleset. In other releases, rules are organized into files where the file name denotes their ruleset. These two ways of specifying the ruleset are both supported by Snowman.

6.4.3 Sensors

Every sensor in the system is represented in the database with its name, IP-address and status. Sensors also contain a list of rulesets applied to them, and active rules within these rulesets will be distributed to the sensors given that the rulesets are also active. A sensor can be configured to be autonomous, meaning it will not communicate with the server. Autonomous sensors can be used in environments where contact between the server and sensor is unwanted or not possible. Rules can be manually exported for these sensors.A sensor can also be the parent of other sensors, allowing users to create groups of sensors with the same configuration. The database also contains an "All Sensors" object, which is the parent of all sensors.

The sensors that are not autonomous are checked if they are active every minute, and the last registered status is stored in the database. This is then used to show the status of the sensor to the user. Each sensor also has a user assigned to it, and this user is used to let the sensors authenticate with the Snowman server.

6.4.4 Comments

The `core` module contains a comment object, which can store comments from users who wish to document their operations, e.g. the user can attach a comment when adding a filter to a rule to describe why the filter was added and which effects it will have on the system. Snowman can also generate comments for important events, such as turning rules and rulesets on or off, editing and deleting sensors, etc.

[6] "The Community Ruleset is a GPLv2 VRT certified ruleset that is distributed free of charge without any VRT License restrictions" - http://www.snort.org/snort-rules/

6.5 Update Module

The purpose of the `update` module is to fetch and read rules and configuration files from external sources, and populate the database with this information. Effectiveness in updates is a key issue, and a number of deliberate optimizations for performing updates, with the goal of saving time by omitting unnecessary operations, are explained in the subsequent sections. To avoid confusion with the module, the term *update-package* is specifically used to refer to update data coming from external sources.

6.5.1 Sources

A *source* is a provider of update-packages. Snowman lets users define sources, which are either external (i.e. online) or local, meaning that the update-package will either be downloaded from the web or read from the local system. All information about each source is contained in the `Source` class. The sources determine where the update-packages are located, and an update-package must always come from a specific source.

External sources contain a URL pointing to the location of the update-package, and alternatively also a URL to the MD5-hash of the package. External sources can be configured to pull updates at regular intervals, contrary to local sources, which must always be updated manually.

A source can be configured to update at regular intervals, or the user can run an update manually on a source. To make the update process more effective, `update` makes use of an MD5 URL which points to a file containing the MD5 hash of the update-package. This hash is stored for every source (if available) and is compared to any pre-existing hash to verify if the system is up to date or not. `update` will only download and process an update-package when it encounters a new hash.

6.5.2 Files

An update-package may contain various files depending on the source it comes from. This section describes the files supported by Snowman. Note that file names and extensions used in this report follows the *de facto* names used in the official Snort rules, but are subject to change. The names and extensions can therefore be manually specified by Snowman users in a configuration file (Section 6.10).

Rule files

A rule file is any file carrying the *.rules* file extension. These files contain raw rule strings to be processed by Snort. A rule string is basically a semicolon-separated list of rule options. Specific rule options used in Snowman are listed in Table 13.

Classification, Generator and Reference Configuration Files

The classification files are identified by their name *classification.config* and contain rule classification strings that are used by Snort to categorize event data. A classification file will normally appear once per update-package, and rarely changes between updates [38]. Generator (*gen-msg.map*) and reference configuration (*reference.config*) files contain strings that specify generator messages and rule reference types, respectively.

Signature Message Files

Alert messages for rules are normally written inline, but can also be specified in a file called *sid-msg.map*, which may also contain rule references.

Data field	Description	Presence
sid	Unique identifier for a rule.	required
rev	Rule revision number.	required
msg	Notification text when rule triggers.	required
classtype	Connects the rule with a specific class of attacks.	required
gid	Generator ID to identify which subsystem generates the event where this rule fires.	optional
ruleset	The ruleset this rule belongs to.	optional
priority	Override default severity set by classtype.	optional
reference	Reference to external resource with information about this attack.	optional

Note: The *presence* column indicates whether a field is required or not by the parser in this system. Information is gathered from the Snort Users Manual [19].

Table 13: Rule options used in this system.

Threshold Files

These files contain filter specifications for specific rules.

6.5.3 Running an Update

An update is either triggered by the user, or automatically at regular intervals, for a given source. Regardless of how the update process is invoked, it follows the same procedure. As mentioned above, sources will often provide an MD5-hash of the update. When this is the case, the hash is stored in the database. Every new update will compare its own hash to hashes stored in the database, and if an identical hash is found this means that the update has been run before, and the update process is terminated. When the hash does not pre-exist in the database, or for sources which does not use hashing, the update-package is downloaded to a temporary directory.

Snowman accepts update-packages in the form of tar-archives[7] containing rule and configuration files, and also supports plain text files containing rules or known configuration entities[8]. Tar-archives are extracted, and folders are traversed recursively to locate files for parsing. In most cases the update-package will contain only minor changes in relation to its predecessor, e.g. only one file is different in the latest package, and parsing all files that has not changed would therefore be a waste. To address this issue and save processing time, the name, path and MD5-hash of every file is stored in the database per source. Every time the update module parses a file, it first creates a hash of the file and compares this with the respective file's hash in the database. If the file is present in the database and the two hashes are identical, the file is considered *not new* and is skipped. The following sections will elaborate on the parsing of files based on file type.

The update module contains one parsing method for each supported file type. Since all information in an update is plain text, these methods use regular expressions to fetch

[7] Files that contain compressed files.
[8] Generators, filters, references etc.

the content. For example, a generator configuration contains a comma-separeted list with a name, description and priority: *config classification: unknown,Unknown Traffic,3*, which are extracted from the string by using a regular expression and put into a RuleClass object which is later stored in the database. Update also contains a method for parsing files containing unspecified entities, where each line is first parsed to decide which entity the data belongs to before it is sent to the correct parsing function for the respective entity.

6.5.4 Rule Parsing

A rule file is read line by line, and each rulestring found is sent to the rule parsing method. The rule string is then matched to a predefined pattern that will verify that the rule is valid and extract the data fields (Table 13).

Rulesets

A rule always belongs to a ruleset, which can be specified in the raw rule string. If the parser does not find this attribute in the rule string, the filename is considered to be the ruleset name.

Filters

As mentioned in Chapter 4, rule filters may appear in rule strings (detection filters) or in external files (event filters). In addition to these filters, Snowman supports a filter type called *threshold*. A rule threshold defines a limit and/or threshold for the triggering of a specific rule, just like an event filter, but can be specified both inline or externally. Rule thresholds are deprecated, and support for this functionality will cease in a future version of Snort [19]. To maintain compatibility with current rules, Snowman supports reading inline and standalone threshold definitions.

All inline filters are removed from the rule strings and transformed into event filter and detection filter objects. This is done to prevent Snort from adding filters to rules when parsing the rule strings, as duplicate filters for the rules will cause Snort to crash. Any filters extracted from rule strings are written as plain text (i.e. as they appear in the original rule string) in the *filters* field of the **RuleRevision** object. This is to ensure that all the original rule options are stored, meaning that no data is lost or discarded.

Suppression

Suppressions can be applied to any rules that might be giving off false positives or generating a lot of unneccessary alerts. There might also be cases where the traffic in question is legal on certain networks or to/from certain IP-addresses. Only one suppression can be applied to a rule on a sensor, but multiple IP-addresses or networks can be listed in the suppression in a comma separated syntax, like [1.2.3.4,4.3.2.1/24]. Rule suppression is always specified in external files.

Snowman features IP-address validation checking to make sure no invalid addresses are entered, by running a regex-match on the address before it is committed. The match will check for a valid IPv4 syntax in the range 0.0.0.0/0 - 255.255.255.255/32. There is currently no support for IPv6, but this can easily be added in future work.

6.5.5 Classification, Generator and Reference Type Parsing

These files contain a sequence of their respective entities (Section 6.5.2), and are all parsed in the same manner. If a given entity is not already present in the database, it is

created and stored. If the database already contains the entity, the existing entry in the database is overwritten.

6.5.6 Saving

Performance testing of update processing (see Chapter 9) showed that Snowman version 0.4 was very slow compared to other software. The reason for this was examined, and found to be a large amount of database calls spread out in the code. Snowman 0.4 saved every rule individually after a change was found and the objects were updated. This principle is demonstrated in Listing 6.3, where the method tries to fetch, alter and save a classification object, or create a new one if not found. This turned out to be a bottleneck, as updates could take hours on low-end hardware. To solve this, we split the methods of `Update` into two new classes: `Parser` and `Updater`, where the first would only parse the incoming data and send this to the latter which gathered all objects and wrote these in bulk when the parsing was complete. As can be seen in Chapter 9, this had a significant impact on performance.

```
1  from core.models import RuleClass
2
3  def updateClassification(type, description, priority):
4    try:
5      # Update existing classification
6      ruleclass = RuleClass.objects.get(classtype=type)
7      ruleclass.description = description
8      ruleclass.priority = priority
9      ruleclass.save()
10   except RuleClass.DoesNotExist:
11     # Add new classification
12     ruleclass = RuleClass.objects.create(classtype=type, ...)
```

Listing 6.3: Updating a classification in Snowman 0.4

As some of the objects refer to other objects in the database through foreign-keys, it is vital to store the objects depending on other objects after the objects they depend on are stored. A safe order to save them has been identified and the objects are saved in the following order:

1. Generators
2. Classifications
3. Reference-types
4. RuleSets
5. Rules
6. RuleReferences
7. Suppresses
8. Filters

For every item in the list above the procedure is approximately the same. First, the objects received are attempted to be extracted from the database. The response from the

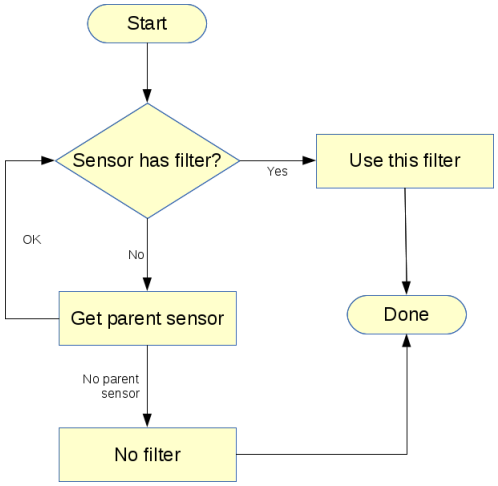


Figure 13: Applying a filter to a rule on a sensor

database lets us identify which objects are new and which are changed. The changed objects can be updated and saved individually before all the new objects are constructed and everything finally gets inserted into the database in one bulk operation.

6.6 Tuning Module

The `tuning` module contains classes that hold information on rule filtering and suppression. Snort supports only one filter of each type per rule. In Snowman, this is translated to one filter of each type per rule *per sensor*, which means that a filter is always connected to both a sensor and a rule. If a filter occurs inline, this filter is regarded as a default filter and will be active on every sensor where the rule operates. This is accomplished by applying the filter to the rule and the *all sensors* sensor, which is the parent of all sensors. When a rule is enabled on a sensor, Snowman will look for any filter set on the rule on the respective sensor or on any of its parents. This means that a default filter can be overridden by setting another filter directly on the sensor. The routine for applying filters to a rule on a sensor is explained in Figure 13.

6.7 Distribute

To distribute the rules from the central database to the sensors in a format that Snort can understand, three major events happen. Each of these are explained in the following three subsections.

6.7.1 Initiating the Distribution

Before a synchronisation can start, the sensors need to know that they should synchronise. There are mainly two events that can cause a synchronisation to occur:

1. Automatic synchronisation based on time
2. Manually invoked trough the web interface on the Snowman-server

In the first case, the client itself knows that it should start to synchronise and therefore starts the authentication/synchronisation procedure with the server. In the second case, the client does not initially know that anything happened. The central server has to

contact the client via the “snowmanclientd” daemon, telling it to initiate a synchronisation.

6.7.2 Synchronisation

When a synchronisation is initiated, the client connects to the “snowmand” daemon on the sentral server and authenticates with it. If the authentication is successful, the server knows that the sensor is who it claims to be and it also knows which rules the sensor should have installed. After this, the synchronisation can start. The first action is to retrieve all the `RuleClass` objects and update the local cache on the client to correspond with the retrieved list from the server. Changed objects will be updated, new objects inserted into the local database and obsolete objects will be deleted. Next, lists of `Generator`, `RuleReferenceType` and `RuleSet` objects are retrieved and the local database is updated in a similar manner as with the `RuleClass` objects.

For the synchronisation of the rules, which is the main bulk of the data needed to be synchronised, a different approach is used. First, a list of the signature-IDs and revisions of the rules that should be present on the sensor are retrieved from the server. If rules in the local cache are not mentioned in this list, or this list indicates that a new version of this rule is available, the rule is deleted from the local cache. Any tuning parameters belonging to this rule are also removed. Next, the client starts to request the rules that it currently is missing from the server. This happens in bulks of 250 rules per request so that no single request becomes too large and time-consuming. For the rules requested, tuning parameters like filters and suppressions are provided if the rules have rule tuning applied on the given sensor. References are also passed along with the rule. All the received rules are then saved to the local database and the synchronisation is complete. The final step of generating Snort configuration files is then started.

6.7.3 Generating Snort Configuration

The generation of the Snort configuration files is a rather simple process. First, the old configuration is cleared out and new files are generated based on the content of the local database. The naming convention of the files is similar to the one SourceFire uses with their rulesets and the format of the files is exactly the same. This is to simplify the visual inspection of the rule files. It does not really matter to Snort which order the configuration files are in, as long as the content can be recognized as Snort configuration.

In addition to the various files created by Snowman, one file is always guaranteed to be generated. That is the “snowman-includes.cfg” file, which includes all the files generated. If Snort includes that file again, all the files generated by Snowman will be included for Snort. Finally, when the configuration has been generated, the snowman-client sends a SIGHUP signal to the Snort process, which makes Snort reload its configuration and start using its new rulesets.

6.8 Graphical User Interface

The Graphical User Interface (GUI) is a template- and AJAX-based web application which uses Django’s template system to implement the View part of the MVC-principle [39]. Each webpage is a template populated through Python code before it is delivered to the web server, and ultimately the requesting web browser. Additional data that needs to be sent to a Web page, e.g. the user interacts with the GUI, is requested through JavaScript and AJAX-calls and then gets added to the already loaded DOM in the web browser.

6.8.1 Structure

The pages of the GUI are organised in the structure presented in Figure 14, which is very similar to the way BHTB organised its front end.

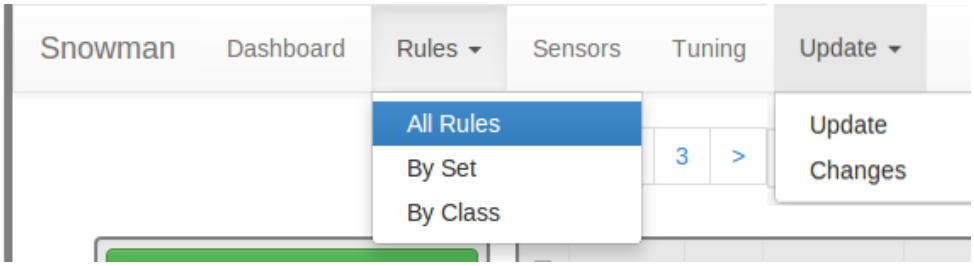


Figure 14: GUI Structure

6.8.2 Design

The GUI is designed with simplicity and minimalism in mind, which is achieved partially by using Bootstrap that has a lot of default no-nonsense design elements. But we also opted for using a lot of primary colors and simple lines to make it aesthetically pleasing.

6.8.3 Overview of Central GUI Features

Content Prefetching

One of the main features of our GUI design is our use of prefetching to speed up loading. Instead of loading absolutely everything into the web browser DOM, like our predecessor BHTB did, we instead initially only load the first page or page elements. This is designed to reduce loading times.

We then silently in the background load in any element the user might want to see next, through AJAX-calls. When the user then interacts with the GUI, the next elements are shown to the user. The next-next elements are also loaded through AJAX-calls so that they are ready. This ensures that the user always has immediate response from any interaction done with the GUI, while keeping loading times to a bare minimum.

Rules Page

This page (Figure 15) serves as the complete list of all rules currently residing in the central database. Because the central database can contain hundreds of thousands of rules, this list is paginated. Pages are loaded using AJAX calls to prefetch them as needed. Each row in the list can be clicked on, which expands an area containing additional information about the rule in question.

We anticipate that the main use of this page is going to be utilizing the search function to find a specific rule or rules and then either retrieve information about them or otherwise manipulate them. Rules can be tuned by checking the checkbox associated with the rule in question and then pressing the button that represents the desired function in the box to the left.

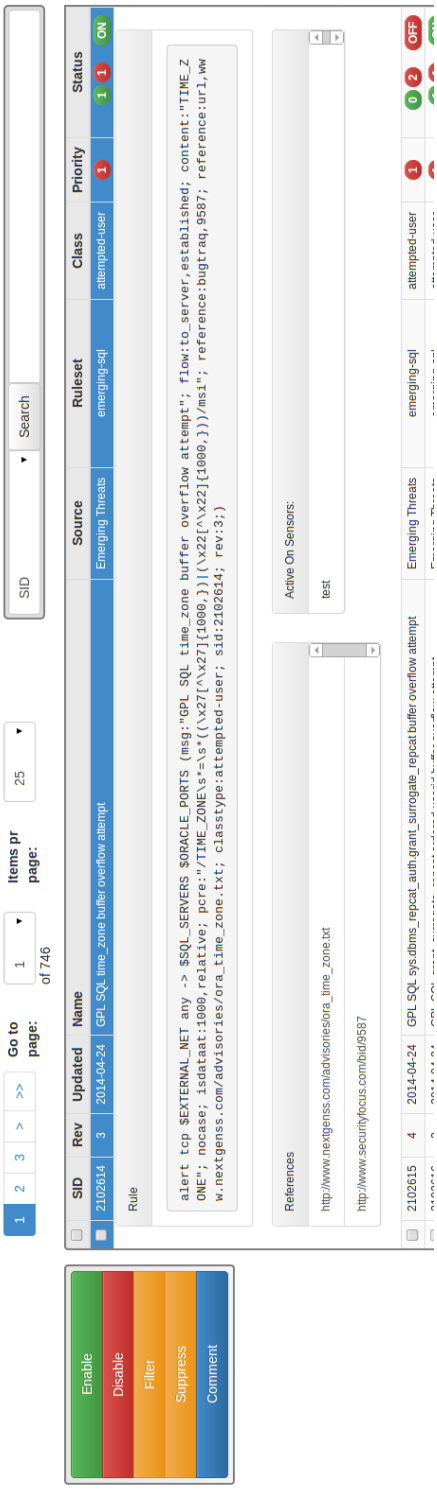


Figure 15: Rules page

Rulesets Page

The rulesets page (Figure 16) consists of a list of all the rulesets stored in the central database. Each row contains meta information about the ruleset, such as the number of rules in the ruleset and so on. A special feature of the ruleset page is that since Snowman allows the user to structure rulesets in a hierarchical tree, the ruleset list will initially only show the tree roots. If the ruleset has child rulesets associated with it, those will be listed once the user click on the ruleset and opens up the additional information area.

The list controls also contain functionality for creating, editing, deleting and reorganizing the rulesets, as well as the ability to turn them on or off either globally or on a per sensor basis. We also have a very similar page for the sensor list, which also allows the user to arrange sensors in a hierarchical tree. Because it is almost identical, it is not detailed in the report.

Updates and Changes

One of the big requests from Avdeling BKI when it came to improvements of features of BHTB was the need to get more feedback from the update process, as BHTB only told you about the status after it was done, leaving the user without further information. To solve this, we added a continuously updating progressbar which tells the user what the system is currently doing and give a rough estimate of the percentage of completion (Figure 17). Another requested feature was the ability to see what was changed during an update. We provide a list for this, which contains each update and its various changes (Figure 18).

The list details any new rulesets, rules or rule revisions an update might have brought with it, as well as a list of rules that have been moved from one ruleset to another. Once the user has finished checking out the changes from one update, the user can remove the update from the list. This makes the list a backlog tool, rather than a perpetually long list or some sort of time limited list (updates in the last 7 days or similar). This gives the user a good overview of what has changed since the last time the user checked the list.

Form Validation and Feedback

One challenge that web frontends always face are forms, as they are a means to alter data in the database and thus become a security challenge. So when utilizing forms, it is important to protect the database against both attackers and user errors. There are three basic steps to implement protection against these security issues:

1. Validate the data provided by the form.
2. Make sure the user cannot input something he/she should not.
3. To never let any raw strings or values from the form be directly inserted into a database query.

The third protection is the most important one as it prevents SQL injection attacks, which are attacks where SQL code is input into the form, letting the attacker carry out his/her own queries, leading to total database compromise. This problem is solved through a technique called prepared statements, which make sure a query does not do anything it is not supposed to before the query is actually executed on the database. This protection is implemented in our system through Django's internal protection mechanisms.

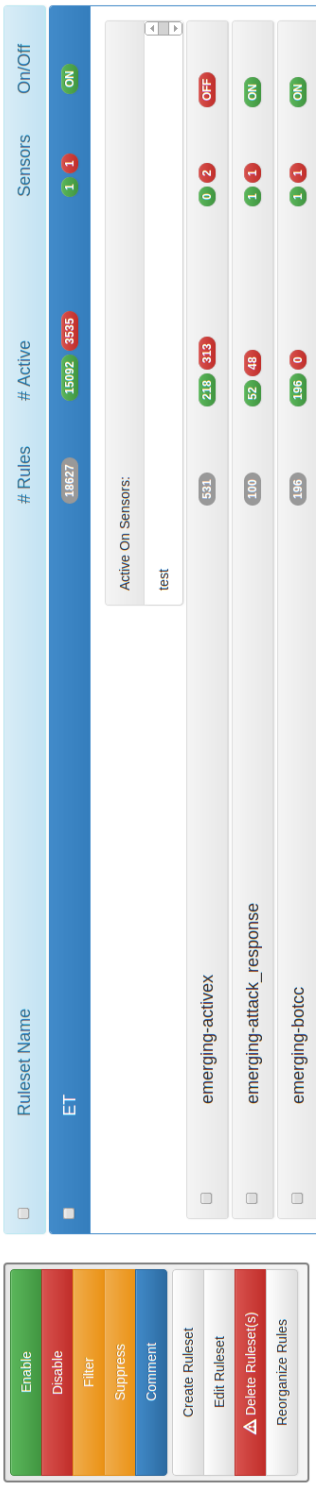


Figure 16: Rulesets page

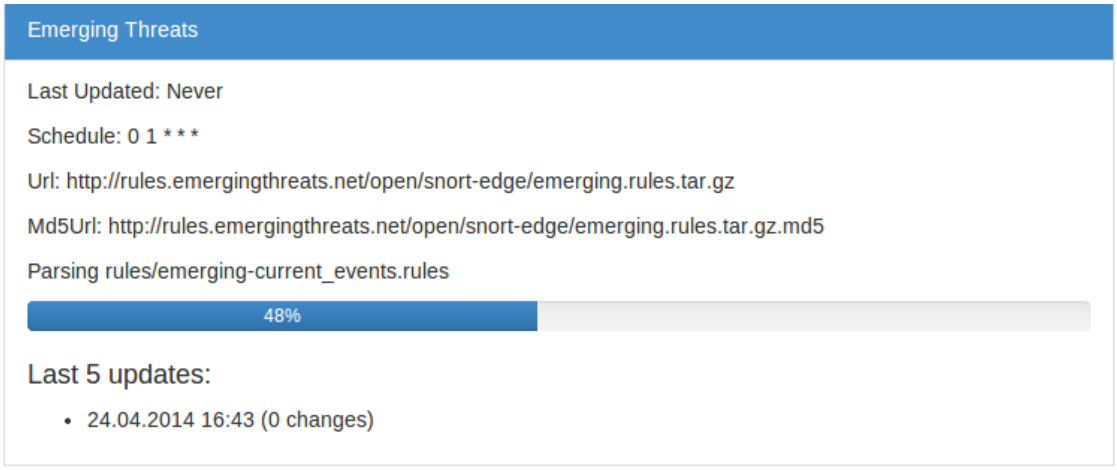


Figure 17: Update progressbar

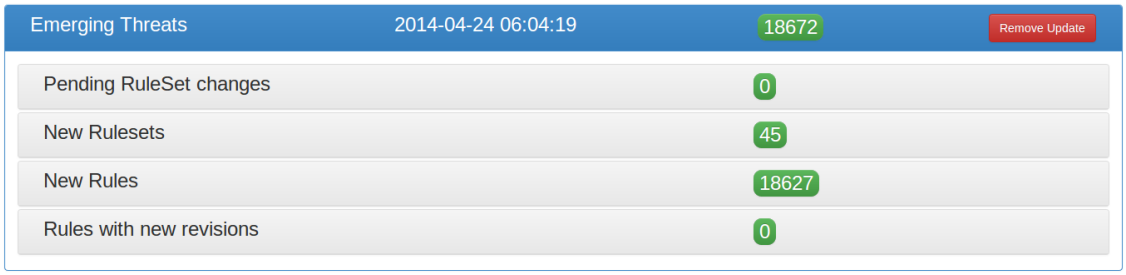


Figure 18: Update changes

The first and second protection is something we have implemented ourselves in both our frontend and our backend. We make sure that the user can never input invalid values into any form by having JavaScript validate the input (Figure 19). So if a user inputs characters where only numbers are allowed, the user will be notified immediately and not allowed to submit the form until it is corrected. And this is done before anything is ever sent from the browser to the server. But if the user has input all valid values, we still validate all the data at the backend before we store anything in the database. This means verifying that any referenced object ID actually exists and making sure the provided IP address is actually valid (within the 0.0.0.0/0 - 255.255.255.255/32 range). If anything is out of place, a message is sent back to the user about it.

6.9 Logging

Complex software needs to log what is happening, to simplify the understanding of what happens when certain things go wrong. It is also needed to document what has been done, in case someone would need to make sure that an update ran or other actions have been taken by the system. As Snowman is a fairly large and complex piece of software, we needed to think about logging of the internal events.

For the logging of internal events in both the Snowman-client, and the Snowman-server, we decided to use the standard Python logging libraries. These libraries allow us to log

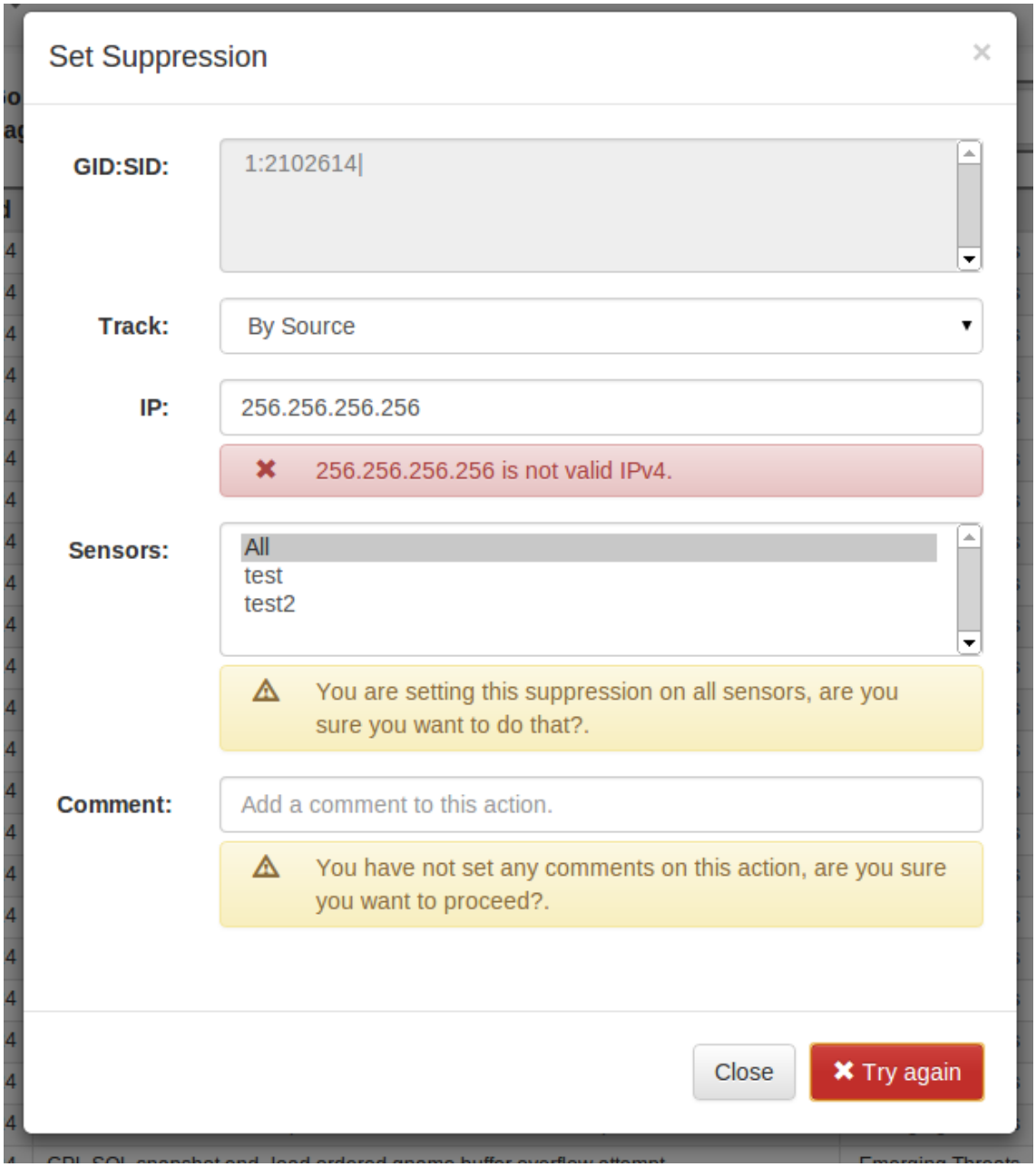


Figure 19: Form validation

events of importance inside our code, depending on how important these events are. The logger would then log information about things like; which modules the events occur in, the time an event happened, and a message describing the event. The levels of logging are Debug, Info, Warning, Error and Critical, sorted in an ascending order of importance.

Handlers can be assigned to the logger, which decides where to store the logged events. Snowman uses one handler by default, which writes events to a file. The configuration file for Snowman decides where this file is stored and which severities are stored. Under normal operation, it is recommended not to log Debug messages, as they are gen-

erally not needed and therefore only use up disk space and may slow down the overall performance.

The Python logging facilities are configurable and flexible. If future needs would require some events to be stored in a database or send messages to users (via SMS for example), the only thing that would be needed to be done is to add a new handler.

6.10 Configuration Files

Snowman's configuration files exist to give users some flexibility by providing a set of options that specify how Snowman operates. The configuration files originated from a need to store database credentials, which of course would differ from system to system. This was expanded to include some options that users might want to set differently depending on their situation, for example if new rule revisions should automatically be active in the production system or not. Snowman requires a configuration file for the server and one for each client, detailed below.

6.10.1 Server Configuration File

The server configuration file contains file paths and other options used for logging and client connections, along with the options described below:

Rule Revisions

Users can specify the number of revisions to store. Snowman will automatically delete the oldest revision when the revision count for one rule exceeds this number. The default value is 2, which is just enough to let users roll back to the previous working revision after an update. Users can also specify whether incoming revisions should automatically become active (i.e. go right into production) or remain inactive until manually activated.

Rulesets

There is a possibility that a rule will be moved from one ruleset to another by a source, for example the Snort rules contain a "deleted" ruleset, which contains rules that were previously part of other rulesets, but are now considered obsolete. Users can also choose to re-arrange rules across rulesets, which may create conflict during updates. This means that some users will prefer that rules automatically moves to the ruleset dictated by the update, while others would like the rules to remain in their original ruleset. This functionality is therefore customizable in the server configuration file.

Rule Filters

Another thing that might cause problems is rule filters. Since there can only be one filter of each type per rule, users must decide whether they want new incoming filters to automatically overwrite old filters or not. In the server configuration file, users can specify that filters should be *a*) always overwritten; *b*) overwritten unless the previous filter was set manually by the user; or *c*) not overwritten at all.

Alert Messages

Rule alert messages can appear both inline and in a separate file, and one of these sources must take precedence if they contain different alert messages. This behaviour can be configured to always use the inline or external specification, or which ever is encountered first.

Rule Caching

Users can specify whether all rules should be cached in memory during an update, even if they are not changed. It might be useful to enable this if the updates contain a lot of filters, as it will reduce database access when updating the filters.

Update Files

Since there is no documented standard for file names and file contents within an update-package, users can specify both a whitelist and a blacklist over which files to parse during an update. If present, the whitelist will take precedence over the blacklist. In the whitelist, users specify a list of filenames together with their expected contents. If a whitelist is specified, Snowman will look for, and parse *only*, the following files:

1. One file containing rule classifications.
2. One file containing generator messages.
3. One file containing alert messages.
4. One file containing rule filters.
5. An arbitrary number of files containing rulestrings.

The whitelist provides the most effective update parsing, as only the files specified are parsed. If a blacklist is used, Snowman will parse every file in the update except files with extensions specified in the blacklist (e.g. ".c" and ".so" files are skipped). This will slow down the parsing, as Snowman has to check if each line in each parsed file matches a known entity, such as a classification or a rulestring.

6.10.2 Client Configuration File

Every client in the system requires its own configuration file. In this file, the credentials of the sensor on which the client operates are stored, along with database and logging options and paths to required directories and files. Parametres which impacts the synchronisation are also included in this file, most notably the request-size, which controls the maximum amount of rules requested in each request. Depending on the resources available on the server, large requests might take too much time to handle, which can make the connection time out. On the other hand, too small requests would increase the synchronisation-time because of I/O-waiting between every request. It is therefore natural to let users define the request size in the configuration file.

6.11 Deployment

The deployment of Snowman is a two-stage procedure. First, the central server needs to be installed and configured. After the server has been verified to be working as intended, the Snowman-clients can be installed alongside of Snort on the sensor. Both the server and the client are delivered packed in Debian-based installation files. These installation files will install the Snowman software, in addition to making sure that the required dependencies are also present on the machine. However, there is one exception to the dependency solving. The server is delivered in both a Debian-flavour in addition to an Ubuntu-flavour. The reason for this is that one of the Python libraries used are currently not in the debian-stable repository, and for Debian it is required to install them via the

Python package source (pip). After the installation, we therefore provide some utilities as a double-check, to be able to verify that all the required tools are present.

As all of the Snowman software is delivered in packages, automation of the installation can be achieved trough the use of standard tools like *aptitude* and configuration tools such as *puppet* and *cfengine*.

6.11.1 Snowman Server

When deploying the Snowman-server, it is wise to do a couple of considerations first. Snowman uses external software for its webserver and database and because Snowman is quite flexible, it is possible to change which technology is used with a small configuration change. The web interface is powered by a Python WSGI [40] gateway and is thus very flexible in which server technology is serving it. The only requirement would be that the web server supports WSGI. As default, the provided configurations use Apache 2 as their web server.

For the database, as the Django database ORM is used, several technologies can easily be utilized. SQLite, MySQL (and thus MariaDB), PostgresSQL and Oracle are all supported. SQLite might work for smaller installations, where simplicity is preferred over performance. The only requirement for an Sqlite database would be access to write a file in the filesystem. However, if performance is required, a dedicated RDMS (Relational Database Management System) would be more suitable. A standard and free alternative here is MySQL.

Installation

When the database technology has been selected and the web server is available, the installation of the Snowman package can be done. The package is delivered in two versions; one for Debian and one for Ubuntu. Both would work on both systems, as the only difference is the dependency list. In the Debian version, the dependency to *python-django* is omitted, as debian-stable currently does not delivere this package in a new enough version. Snowman requires Django version 1.6 or newer, which is available in the ubuntu-repositories. For debian, *python-pip* is recommended to install Django, as the pip repositories supplies version 1.6.

Configuration

After the installation of Snowman, it needs some configuration to be able to operate. The main configuration file for Snowman is "/etc/snowman/snowman.conf", and all the parametres which are required to be configured are present in that file. The configuration file is well documented within, and all parametres are explained. The parametres that are required to be updated in the file is:

- *Database settings:* The settings determining which type of database to use, where to find it and the authentication details.
- *Host names:* The names listed at this configuration option determines which hostnames the Snowman web interface would answer to. Any request to Snowman using other hostnames than the ones listed would result in an error.

To configure the web server, an Apache 2 configuration file is provided in "/etc/apache2/sites-available/snowman.conf". For Apache 2 to work as the web server for Snowman, an "apache2 wsgi-gateway" needs to be installed. This is also widely available in most Linux

repositories under the name “libapache2-mod-wsgi”. A symlink to the configuration-file should be created in “/etc/apache2/sites-enabled/”, after which Apache 2 needs to be restarted to finalize the webserver setup.

When the configuration changes are complete, the command “snowman-initialize” should be used to generate the database schema and initialise the database with some initial data. The output of this command will also signal if the database settings are correct, as it cannot create the initial data without access to the database.

Verification

When Snowman has been configured, it is time to verify that the configuration is valid. There are five things that need to be verified:

- Database availability.
- Access to logging locations.
- Access to scratch-location on the filesystem.
- Availability of the web interface.
- Availability of the socket used by clients to download updates.

The command “snowman-test” is provided in order to have an automated way to test all these parametres. If everything is OK, a message will state so and the program returns 0. If any of the above bullet points fails their corresponding test, a message is displayed and the test returns 1.

6.11.2 Snowman Client

The deployment of the Snowman-client is a simple three-step approach:

1. Install the snowman-client package
2. Update “/etc/snowman/snowman-client.config” with database settings, address for the central server, sensor name and sensor password.
3. Add the sensor to the Snowman-server, for example trough the web interface.

To be able to verify that everything is as it should be, a tool is provided to test that the client is properly installed and configured. The command “snowmanclient-test” will verify all these points:

- The required libraries are present.
- The configuration specifies a database that is accessible.
- The location for creating Snort rule and configuration files is writable.
- A contact to the central server can be established.

snowmanclientd

There is a daemon supplied with the Snowman-client, named “snowmanclientd”. It is responsible for allowing the central server to make contact with the client. It is set up to automatically start at bootup, but it is also possible to control via the ordinary linux command “service”; “service snowmanclientd start|stop”.

7 SOFTWARE TESTING

7.1 Strategy

The overall strategy of the project when it comes to testing has, first and foremost, been to have regular meetings from Avdeling BKI where we conducted demonstrations of the software. This has lead us to either confirm elements as feature complete or to create a list of elements to either fix or improve. We have also not had any specific regimes when it comes to bug fixing, we have mostly just handled them when they have been discovered them. We could describe our software testing strategy as ad-hoc based, as it is quite common in agile projects. Our main focus was on the performance testing.

Unit testing has not been used extensively in this project, except to test that data from an update is correctly inserted into the database and for some of the functionality in the web interface. Source code listings of all unit tests in this project can be found in Appendix G.

7.2 Description of Problems

7.2.1 Rule Insertion

Snowman extracts various types of data from the rule string and inserts this into the database in different tables. Since an update can contain thousands of rules, it proved to be very difficult to effectively verify that all rules were correctly inserted into the database. To address this, a test suite was set up executing a dummy update with an example rule string containing all possible options. These tests were also very useful during the refactoring of the `update` module, as the new code could be quickly verified. Listing 7.1 demonstrates a test case for rule insertion.

```
def test_updateRule(self):
  # Insert the rule
  self.update.updateRule(self.rulestring, "example.rules")

  try:
    # Verify that all related objects exist
    rule = Rule.objects.get(SID=1, active=True, priority=10)
    generator = rule.generator
    ...
  except Rule.DoesNotExist:
    self.fail("Rule does not exist")
  except Generator.DoesNotExist:
    self.fail("Generator does not exist")
  except ...

  self.assertTrue(rule.active==True)
  self.assertTrue(int(rule.priority)==10)
  self.assertTrue(generator.GID==1)
```

Listing 7.1: Example test case

7.2.2 List Pagination

One problem that has plagued the software throughout the development has been the pagination functionality in the GUI. Due to it using AJAX to fetch other pages in the background, if we changed pages so fast that the system could not catch up, it would just end up never loading anything due to AJAX's asynchronous nature[1]. This has been remedied by using some logic to check for outstanding AJAX calls but a better solution would be to implement some other mechanism or framework for pagination. The remedy has also put some extra latency on the rule list, as waiting for the AJAX calls might take some time.

7.2.3 The Ever Changing Number of Changes

One smaller bug that popped up was on the Changes page where the number of new rulesets, rules and rule revisions there has been in the latest updates are displayed. Initially, only the rule revision was tied to the Update object and we then traversed from the revision object to the rule object and counted how many revisions that rule had, in order to determine if the rule was new[2]. But using this method led to an undesired outcome when a "new" rule gets a new revision in a later update. As the count for revisions is done in real time, this rule will suddenly not be listed as new anymore in the old update.

This problem was solved by tying all relevant new objects to the Update object and then just listing those relations. The downside of this is a lot more relations to store in the database, but the Changes view was one of the more important features so it had to be done this way.

[1] The code does not wait for AJAX to finish before moving on.
[2] A new rule will always only have one revision.

8 PERFORMANCE ANALYSIS

8.1 Theoretical Analysis

The purpose of the theoretical performance analysis is to examine the most important features of Snowman and similar software, and form opinions on the expected performance of software compared with it. The theoretical analysis is used to create concrete benchmarking tests presented in Section 8.2. The software analysed is BHTB, Snortmanager and two versions of Snowman, versions 0.4 and 0.5.

8.1.1 Effectiveness of Update Processing

As described in Chapter 6, Snowman extracts several rule options from the rule string during parsing. These options will either create or update objects related with the rule, such as ruleset, rule message and classtype. These logical relations add to the complexity of the database transactions during the update process, as multiple database tables are involved. BHTB and Snortmanager, on the other hand, write rule data to a single table. It is therefore expected that BHTB and Snortmanager will generally exhibit a more effective update processing.

Incremental updates

One of the expected strengths of Snowman lies in processing incremental updates, i.e. updates containing only a small number of changes. It is important to measure performance for incremenal updates, as most real-life updates usually contain between 10 and 100 changes. For example, ten updates from 20.03.2014 to 17.04.2014 contained on average 36.8 new and 53.9 updated rules [41]. Snowman will effectively handle these updates, as it stores the checksum of the files in the update. Storing the checksum for every individual file[1] is unique for Snowman. It allows Snowman to completely ignore files without changes, which means that only the files with rule changes are parsed. This is particularly useful as the current official Snort rules contain over 100 files[2]. It is therefore expected that Snowman will efficiently process updates that contain relatively few changes.

8.1.2 Effectiveness of Rule Distribution

The rule distribution, or synchronisation, is a feature of Snowman and BHTB, where rules are distributed from the server to the sensors. Since Snowman and BHTB use different strategies for distribution (Snowman uses a local client database as described in chapter 5.1.1 while BHTB pushes all files out to the sensor at each synchronisation), it is previsioned that Snowman will perform better because of the fact that it only transfers new changes rather than everything, every time. Also, since BHTB distributes everything, every time, it is expected to have a flat performance curve, since the timing will be dependent on the bandwidth rather than processing power. Snowman's server-client connection is also more complex than the one of BHTB. Because of this, a longer processing time for setting up the connection is expected.

[1] Checksum is stored only for files that are parsed.
[2] As of May 2014.

8.1.3 GUI Latency

A major complaint from Avdeling BKI was that they experienced a high latency in BHTB's web component, that they had to wait long periods for it to load data. We have attempted to remedy this problem in Snowman by not loading absolutely everything into the GUI every time, like BHTB does. It is expected that this will reduce the latency in the GUI, as we would be processing and displaying only 100/28000 rules as opposed to all 28000 rules.

8.1.4 Sublinearity in Performance

We're expecting all of the software to have a linear performance curve, or better (sublinearity), as an upwards pointing performance curve is a sure sign that something has been programmed in a wrong way. If performing one task takes 1 second, performing 10 tasks should take 10 seconds or less.

8.2 Practical analysis

Since Snowman is similar to BHTB and Snortmanager, it was important to benchmark their performance in various formal test cases in order to highlight the strengths and weaknesses in each software solution. The purpose of the tests is to reach final conclusions on key questions regarding the performance of Snowman, and to point out where the difference between the three software packages lies.

To get a picture of how the software performs on different systems, it was tested on three architectures (table 14). To minimize error, each test was run five times where the mean value of the runs is considered a valid test result. All tests can be reproduced with the test procedure documentation in Appendix F.

8.2.1 Definitions

The testing is split up into *test suites*, each containing a *problem* in the form of a question. The problem forms a clear description of what is to be tested, and how results should be evaluated. A test suite also contains one or more *test cases*, which exist to address the problem. Test cases represent individual tests, and describe the input data and constraints. Test cases are carried out on the different systems, and results from each test case are referred to as the *test results*.

8.2.2 Environment

All tests are carried out on three systems, named *S1*, *S2* and *S3* (Table 14). These systems differ in processing power, available memory and storage technology, and the results will indicate what impact these factors have on software performance. The performance testing in this report is also prone to error, as tests might get disturbed by other background processes etc. Testing on three systems gives results with a higher confidence, as results are checked aginst each other for inconsistencies. When an inconsistency in results is found, the test is run again to see if the inconsistency prevails.

8.2.3 Test suites

Even though Snowman, BHTB and Snortmanager share the same goal of being a rule management system, it is not immediately obvious how they can be tested against each other to compare their performance. The first step in the performance testing was to

System	Processor type	Memory	Storage
S1	Intel® Core™ 2 Solo, 1.4GHz	4GB RAM	5K RPM HDD
S2	Intel® Core™ i7, 6x3.2GHz	12GB RAM	10K RPM HDD
S3	Intel® Core™ i5, 4x3.2GHz	16GB RAM	SSD

Table 14: Test systems.

identify the areas where the three software packages share, to a certain extent[3], the same functionality. In the test planning, considerations have been made to ensure that the tests hold an acceptable degree of fairness (i.e. tests are designed so that none of the programs enjoys any improper advantages or disatvantages), and that the tests are pertinent to the daily operation of the software and the Snowman requirement specification.

Snowman, BHTB and Snortmanager all have the functionality to download and process an update from an external source. The first group of test suites presents different test cases to benchmark how each software performs in update processing.

8.2.4 Update processing tests

The purpose of the tests in this section is first and foremost to test how quickly the software can process an update, and if the processing time is sublinear regarding the update size. It is interesting to test for sublinearity as this may help point out programming errors. To get a complete picture of the performance, the tests are set up with different constraints such as update size, number of files and number of changes from one update to the next. Tests numbered 1-1 through 1-4 use an artificially generated update package where all rules contain the same data[4], as seen in Listing 8.1.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS \
(msg:"INDICATOR-COMPROMISE c99shell.php \
command request - security"; \
flow:to_server,established; urilen:<50; content:"act=security";\
fast_pattern:only; http_uri; metadata:service http; \
classtype:policy-violation; sid:<int>; rev:2;)
```

Listing 8.1: Test rule string.

Test 1-1

Problem: How fast can the software process an update?

The purpose of this question is to determine the average processing time per rule. Results are ranked from short to long processing time, where shorter is better.

Test cases:

To address this problem, five test cases were introduced. The tests contained an artificially generated update with 1, 10, 100, 1000 and 10000 rules, respectively. Rules were split into files each containing ten rules. Each test case represents one update, and the names denote the number of rules in the following structure: *SYN-number of rules*. The test was also carried out with a realistic update package, *SF-28064*.

[3] Snowman, BHTB and Snortmanager have been built to tackle many of the same challenges, however, the actual implementations may differ severely. For an extended walk-through of differences in implementation, refer to Chapter 2, and Chapter 10.

[4] Only signature IDs are different. The rules contain a message and a classtype.

Test case	In test	Description
SYN-1	Test 1-1, 1-2	Blank database.
SYN-10		
SYN-100		
SYN-1000		
SYN-10000		
SYN-1000-B	Test 1-3	All rules in one file.
SYN-1000-C	Test 1-4	Pre-populated database. No changes in update.
SYN-1000-D	Test 1-5	Pre-populated database. Revision change for all rules
SF-28064	Test 1-1	Blank database.
SF-28064-B	Test 1-6	Pre-populated database. Update has one rule change.

Note: Rules are split into files with ten rules per file unless otherwise specified.

Table 15: Test cases in update processing.

Test 1-2

Problem: How strong is the linear correlation between processing time and the size of an update?

This question rises from the requirements specification of Snowman (Chapter 3), where it is stated that Snowman shall exhibit a sublinear increase in processing time in correlation to the number of rules in the update (where the number of rules depicts the update size). Results are ranked from strong correlation to no correlation, where stronger is better.

Test cases:

This test uses the same test cases as Test 1-1.

Test 1-3

Problem: Does raising the ratio between files and rules increase processing time?

Updates can come with the rules either gathered in one file, or distributed over several files. This question is asked to examine the difference in processing time for these two scenarios. Results are ranked from no difference to great difference, where less difference is better.

Test cases:

This test compares the *SYN-1000* test case from Test 1 with a test case with the same number of rules in *one* file (*SYN-1000-B*).

Test 1-4

Problem: Is the processing time shorter when the update contains no new rules?

This is a test for update processing when the update is not new (i.e. it has been completely processed by the system before). To minimize system load, it is essential that already processed updates are (in practice, they should be) neglected by the system. Results are ranked from shortest to longest processing time, where shorter is better.

Test cases:

This is a compound test where a test case is run two times in succession. The *SYN-1000* test case from Test 1 is run first to populate the database, before it is run again (*SYN-1000-C*).

Test 1-5

Problem: Are the processing times for two updates identical when only revision numbers are changed between the updates?

This question measures how one small change in all rules affects procesing time. This test runs the *SYN-1000* update to populate the database, and then the *SYN-1000-D*, which contains a revision number change for all rules. Processing time is expected to be faster for the second update, since the database will already be populated with the majority of the data. Results from this test will be ranked by the difference in processing time between the first and the second updates, where greater difference is better.

Test 1-6

Problem: Are the processing times for two updates identical when only one rule has changed between the updates?

This question seeks to determine how the software handles an update when there is only one change from the previous update. This is an important factor to consider, since an update usually contains only a small amount of changes from previous updates. Results are ranked from short processing time for second update to no difference in processing time, where shorter time is better.

Test cases:

This test suite contains two test cases with an update from a real source. The *SF28061* is run first to populate the database before *SF28061-B* is run, which contains one change (one of the rules has an updated revision number).

8.2.5 Rule distribution tests

These tests use exactly the same test suites as update processing tests, the only difference being that an update from a source is replaced by synchronisation to a client.

8.2.6 Interface loading tests

These tests will evaluate the speed of the user interface by listing rules and rule sets in a web browser. Tests are timed by using the development tools of the Chrome browser, which contains a simple timer. Both tests use the SYN-10000 package.

Test 3-1

Problem: How quickly does the user interface list rules?

Test cases:

This test uses one test case, called *list rules*, where 100 rules are listed in the user interface.

Test 3-2

Problem: How quickly does the user interface list rule sets?

Test cases:

This test uses one test case, called *list rule sets*, where all rule sets are listed in the user interface.

9 RESULTS OF PERFORMANCE TESTING

This chapter presents the results from the performance testing. Results are mean times calculated from five sequential test results, in microseconds. Test cases and systems are listed in rows, with one column per software. See Appendix H for a complete listing of results. Results for interface performance, update and synchronise are presented in Tables 16, 17 and 18, respectively.

Test case	System	Snowman v 0.5	BHTB
List rules	S1	2508	924.561
	S3	1045.950	657.408
List rule sets	S1	11306	39432
	S3	4676	10524

Table 16: Test results for interface performance.

Test case	System	Snowman v 0.4	Snowman v 0.5	Snortmanager	BHTB
SYN-1	S1	1058.179	1415.484	228.617	125.236
	S2	622.640	804.681	93.403	65.281
	S3	151.946	196.160	36.960	18.175
SYN-10	S1	2289.518	1427.712	265.275	148.323
	S2	1668.142	808.218	107.554	72.025
	S3	327.655	195.159	37.569	19.711
SYN-100	S1	17519.385	1568.224	1251.275	185.970
	S2	12197.192	865.496	139.005	84.913
	S3	2356.610	343.033	159.113	25.024
SYN-1000	S1	167645.267	2568.235	9470.460	568.333
	S2	114617.561	1098.159	460.394	158.351
	S3	22830.358	1823.785	1360.276	60.313
SYN-10000	S1	1795175.128	14179.095	92428.555	3606.043
	S2	1162468.835	5761.811	5105.158	910.172
	S3	228038.066	16804.950	13523.887	440.334
SYN-1000-B	S1	147170.285	2508.530	1836.396	445.476
	S2	113901.478	1245.906	496.204	157.817
	S3	20367.838	404.022	228.044	49.908
SYN-1000-C	S1	296.272	309.576	1763.480	7.816
	S2	186.215	169.476	829.230	4.543
	S3	56.152	54.824	205.956	1.310
SYN-1000-D	S1	1371.722	1431.436	64100.000	563.682
	S2	9682.763	748.258	31999.710	85.289
	S3	16998.276	2022.763	10716.689	83.275
SF-28064	S1	9175886.000	63961.545	43391.700	5638.424
	S2	3817584.070	26875.246	21127.828	1715.521
	S3	1267483.848	12522.786	8039.046	1017.561
SF-28064-B	S1	28200.953	9239.670	239878.897	5606.518
	S2	22673.225	5489.009	80009.077	2396.780
	S3	5260.553	1707.588	63626.550	1004.424

Table 17: Test results for update processing.

Test case	System	Snowman v 0.5	BHTB
SYN-1	S1	1645.639	1501.465
	S2	1503.815	1467.612
	S3	371.537	3.300
SYN-10	S1	1828.779	1489.364
	S2	1752.954	1472.808
	S3	453.362	3.462
SYN-100	S1	3975.408	1489.645
	S2	4036.130	1473.355
	S3	1094.713	4.492
SYN-1000	S1	24921.176	1771.004
	S2	25374.218	1503.511
	S3	7759.566	15.132
SYN-10000	S1	239641.965	1839.841
	S2	251607.202	1593.889
	S3	16862.160	121.677
SYN-1000-B	S1	24916.964	1522.719
	S2	25228.116	1481.411
	S3	7252.062	4.156
SYN-1000-C	S1	4265.931	1775.423
	S2	4297.144	1515.184
	S3	2278.760	1097.711
SYN-1000-D	S1	6523.340	1775.664
	S2	6544.584	1513.944
	S3	11642.046	1064.944
SF-28064	S1	112509.497	1758.258
	S2	105555.617	1584.972
	S3	40004.218	1062.806
SF-28064-B	S1	26362.894	1754.601
	S2	23033.204	1584.696
	S3	8839.964	1117.693

Table 18: Test results for sensor synchronisation.

10 ANALYSIS OF PERFORMANCE TESTING RESULTS

10.1 Effectiveness of update processing

This section discusses the most important results from the update processing tests. Table 19 displays the average processing time per rule calculated from the synthetic update containing 10000 rules and the realistic update containing 28064 rules. As it can be seen, BHTB is significantly faster than the two other programs on all our test machines. It came as a big surprise that BHTB performed so well in the update tests, as Avdeling BKI had listed poor update performance as one of its major defects. The results are probably best explained by the very simple nature of BHTB's data structure. Also note that these tests might not reflect the reality that Avdeling BKI faces, as they are run in a minimalistic environment without any sensors or other database-objects that might affect BHTB's performance in a real environment. Snortmanager also performed well in the tests, which is expected as it has a very similar data structure to BHTB in that it is very simple.

Test	System	Snowman v 0.5	Snortmanager	BHTB
SYN-10000	S1	1.4179	9.2429	0.3606
	S2	0.5762	0.5105	0.0919
	S3	1.6805	1.3524	0.0440
SF-28064	S1	6.3962	4.3392	0.5638
	S2	2.6875	2.1128	0.1716
	S3	1.2523	0.8039	0.1018

Note: All times in milliseconds.

Table 19: Average processing time per rule.

Snortmanager starts to struggle when more files are introduced into the tests. As it can be seen in Table 20, there is a great difference between tests *SYN-1000* and *SYN-1000-B*, where we have the same number of rules, but spread over more files. This problem does not seem to exist in either Snowman or BHTB. None the less, both Snortmanager and BHTB was much better than Snowman v0.4 in these tests, which led us to completely rework our update program workflow. Our problem was that for every rule, we were writing its data to the database and then moving on to the next rule. This caused a lot of busy-waiting for the database. We can see that the times accelerated a lot on the machine with the lower-end hard drives, so the problem also manifests itself more when the hard drives cannot keep up.

We proceeded to rewrite the code so that all the rules are parsed into memory first and then is committed into the database in one big query. As we can see in table 21 we improved Snowman's performance in v 0.5 significantly.

Test	System	Snowman v 0.5	Snortmanager	BHTB
SYN-1000	S1	2568.235	**9470.460**	568.333
SYN-1000-B	S1	2508.530	1836.396	445.476
SYN-1000	S2	1098.159	460.394	158.351
SYN-1000-B	S2	1245.906	496.204	157.817
SYN-1000	S3	**1823.785**	**1360.276**	60.313
SYN-1000-B	S3	404.022	228.044	49.908

Note: All times in milliseconds.

Table 20: Comparison of processing time for many files vs one file.

Test case	System	Snowman v 0.4	Snowman v 0.5
SYN-1	S1	1058.179	**1415.484**
	S2	622.640	**804.681**
	S3	151.946	**196.160**
SYN-10	S1	2289.518	**1427.712**
	S2	1668.142	**808.218**
	S3	327.655	**195.159**
SYN-100	S1	17519.385	**1568.224**
	S2	12197.192	**865.496**
	S3	2356.610	**343.033**
SYN-1000	S1	167645.267	**2568.235**
	S2	114617.561	**1098.159**
	S3	22830.358	**1823.785**
SYN-10000	S1	1795175.128	**14179.095**
	S2	1162468.835	**5761.811**
	S3	228038.066	**16804.950**

Note: All times in milliseconds.

Table 21: Comparison of Snowman v 0.4 and v 0.5.

While we still did not accomplish the performance goals set in Section 1.5.2, considering the difference in complexity and structure, we assess the difference as acceptable. The results in these tests are a very good indicator that performance testing during development is a very useful element in a development project.

10.2 Effectiveness of rule distribution

As we expected, BHTB had a very flat performance curve in this test, but not exactly flat. There is a small increase in time spent when the number of rules goes up, as we can see

in Table 22, meaning there is some degree of processing time influencing the results.

Test case	System	BHTB
SYN-1	S1	**1501.465**
	S2	1467.612
	S3	3.300
SYN-10	S1	**1489.364**
	S2	1472.808
	S3	3.462
SYN-100	S1	**1489.645**
	S2	1473.355
	S3	4.492
SYN-1000	S1	**1771.004**
	S2	1503.511
	S3	15.132
SYN-10000	S1	**1839.841**
	S2	1593.889
	S3	121.677

Note: All times in milliseconds.

Table 22: BHTB Syncronisation performance has a flat curve.

Still, considering the tests on S1 and S2 were carried out on the same network and sensor, and S3 were carried out on a completely different network and sensor, it is apparent that distribution from BHTB is mostly determined by how fast the data can be moved from the server to the sensor. The same difference in testing environments is apparent in the Snowman tests (Table 18). S1 and S2 have very similar results while S3 probably benefits from its different hardware and network setup. This leads us to conclude that the performance of distribution is dictated by the bandwidth and by the hardware of the sensor, and not so much by the hardware of the central server. However, there are no indications of a pure performance gain over BHTB as we expected by going for a distributed model with a change based distribution protocol. But we are seeing a significant reduction in time spent distributing when there are only updates and not a massive batch of rules. While BHTB uses roughly the same time on tests SF-28064-A and SF-28064-B, we use about 80% less time on test B than on test A (Table 18). This gives merit to the model we picked for our distribution architecture and with a little more performance optimisations, the overall times are likely to improve further as well.

10.3 GUI latency

These tests show that we have not been able to produce an improvement over BHTB when it comes to displaying rules, which comes clear when we compare the numbers involved (Table 16). Snowman are only fetching and displaying a fraction of the number

of rules that BHTB does[1], yet we spend about twice as long on it. We strongly suspect that the cause of this is a combination of our more complex data structure, coupled with a complex template framework such as Django. A small optimization was attempted by enabling a setting that lets Django prefetch related objects from the database and that did improve the results to some extent, but not enough to be noticable.

10.4 Sublinearity in performance

Our tests show that all the systems were more or less exactly linear, which means that none of the systems are suffering from bad programming that manifests itself exponentially. Table 23 shows the linear correlation between the test results for the synthetic updates with 1, 10, 100, 1000, and 10000 rules. The first dataset is the test results and the second dataset is $\{1, 10, 100, 1000, 10000\}$. As can be seen, the performance is close to linear for all software across all systems.

System	Snowman v 0.5	Snortmanager	BHTB
S1	0.999945	0.999999	0.999695
S2	0.993996	0.999627	0.999931
S3	0.999999	0.999999	0.999983

Correlation $\rho = \frac{Cov(X,Y)}{\sigma_x \sigma_y}$

Table 23: Empirical linear correlation coefficients for update processing.

10.5 Conclusions

Snowman did not outperform either BHTB or Snortmanager in any of the tests, but through the test we managed to discover a big performance bottleneck in Snowman's Update module that we managed to fix in time. The fix resulted in a great improvement in performance. The extra latency in Snowman is likely due to its more complex data structure surrounding rules, which means more processing and data movement is needed for many operations. The project has not exhausted all possible optimization possibilities and techniques, so there is room for further improvements in future work. These tests stand to point out that performance testing in software projects can reveal performance problems early enough in the development to correct them.

[1] About 100 to 28000.

11 CONCLUSIONS

11.1 Results

This project set out to develop, from the ground up, something that were to be better than a prototype already in use. And we have produced something that does indeed works and, in many cases, better than it's predecessor. While not all the features we outlined in our goals ended up in the latest version, they can be added in future works with relative ease. Even though our performance tests show that we did not reach all our goals when it came to making Snowman more efficient than BHTB, we still feel the results are satisfactory. The added latencies in Snowman can be acredited to the more complex data structure which results in more data needing to be processed and moved to/from I/O devices.

The performance tests in chapters 8 to 10 also revealed that actually conducting these tests in a software development project is a very good idea, as we discovered a bottleneck in our Update module and managed to fix it and improve our performance significantly. Some aspects of Snowman are still not within acceptable performance times, but the optimization done on the Update module brought it into an acceptable range for that module. If similar optimizations are implemented in other modules in future work, we are confident that Snowman can perform well across the board. While we have not conducted tests on the matter, we believe Snowman does not have the same scalability problems that BHTB seems to have when adding more sensors to the system. This has to do with our distributed model, removing the bottleneck of being dependent on a central system for the distribution of rules and removing the bandwidth strain of distributing everything, every time.

We also created a GUI that offers more functionality than what the GUI of BHTB does. It is not perfect by any stretch of the imagination, but it features more essential features such as form validation and user feedback. The project also reveals that using frameworks such as Django to create an object abstraction on top of the database can really let us create a lot of functionality in a shorter amount of time, than if we had to work directly on the database.And perhaps the most important conclusion, the members of the project group have learned a great deal when it comes to software development projects, especially when it comes to working with larger enterprise frameworks such as Django. We also learned a great deal when it comes to time management in larger project such as this, and about IDS systems in general.

11.2 Future Work

At the start of the project we were ambitious with our project goals and Avdeling BKI had a long list of functionality they wanted implemented. While most of the functionality made it to our latest version, some where left for future development. These functions include the ability to comment rules, the ability to write new custom rules and to have some way to check for errors in the rules before they are distributed. There are, however, no problems adding these features in future work, as we've strived to make the program

modularized and accepting of additional modules.

To solve the functionality of checking for errors in rules, we discussed early on if something like Dumbpig [42] could be used. It gives feedback on rule syntax as well, so it could also be used to check rules that have been written by the user for mistakes or errors. There is also more optimization work that could be done to speed the program up, especially when it comes to fetching lists of rules. Rule distribution will also benefit from a similar optimization as done with rule updating.

The GUI could probably also benefit from more work, as it has a few quirks and odd bugs. It would probably be beneficial to find some other way of doing pagination, as the implementation in the current iteration is not optimal. There are also some cosmetic improvements to be done to tighten up the visual aspect, on some of the pages it can become hard to differentiate items once we've navigated and opened too many elements on the screen. We also compiled an improvements list from Avdeling BKI on some of the improvements they would like to see in the future, which is available in appendix I.

11.2.1 Abandoned Performance Tests

Some performance tests were not conducted in this project. These tests include real-world tests where the programs are tested in a real detection environment to give a better understanding of software performance in production systems. It would also have been interesting to run the distribution tests to multiple clients to verify that running parallell synchronisations has an effect. This project has also not tested the resource usage of the software, such as CPU and memory.

11.3 Conclusion

The project group has developed a system that works as intended and solves the majority of the features requested of it. The final software solution is ready for deployment, and should simplify the management of Snort signatures while being an effective and scalable system. The group believes that the software can be made useful in the battle to keep the internet safer, but only time will tell if it will be adopted by the security community. One of the main points of this report is that achieveing high performance in software is very difficult when the expectations of functionality is high. Performance and functionality is well-balanced in the developed software, even though test results show that it can be slower than similar software. This project has also highlighted that performance testing during development is beneficial, as it reveals bottlenecks in the software. Because of this, we were able to implement optimizations that drastically improved the performance of one of our system modules.

12 Group Evaluation

This chapter presents our subjective views of the project.

12.1 Introduction

In the course of the project, the group has worked well as both a team and as individuals, drawing from each other strengths and expertise. We have created something that could very well either live on in service of the information security community or inspire others to further improve or recreate.

12.2 Organization

The group was given an interesting task of to look at an existing software prototype, and build a completely new system with added functionality and better performance. In addition, Snortmanager, which is a very similar project, had already been conducted at Gjøvik University College. It was therefore important not only to create a solid system, but to develop a software that is clearly different than any existing solutions. This resulted in a thourough design-phase that laid a solid foundation for the development.

While we had a designated group leader, we never really got in a situation where we needed one. Most of our decisions or problems were solved as a group through discussion and a democratic approach and that worked well for our group. It seems like we succeeded when it came to our project plan, which we have more or less followed throughout the entire project. The development cycle lengths and intermittent meetings have allowed us to be very flexible and agile in our development, as well as being more or less according to schedule. Spending some time planning everything before we started programming gave us a good head start and even saved us some time writing this report as we had a lot of its components already made. When we completed the initial planning phase, we held a “project decision meeting” with Avdeling BKI where we introduced the system models we had made so that Avdeling BKI could decide what they wanted or not. We felt this was a good idea, as we could then get some valuable feedback before we started programming.

Our approach to the development cycles was mostly as described in section 1.7.1 we conducted synchronisational meetings once or twice a week to chart out the course, and keep up to date with what the others were doing. But for the most part, because we had divided the software into modules, we worked individually. This worked out well, as the group members had different schedules and needed the flexibility. For the thesis, we set a good month and a half of our project time for it and we think it paid off, especially as we expanded it to include the performance testing part. We also divided the writing part into three draft cycles, so we could get valuable feedback from our supervisor and Avdeling BKI.

12.3 Work distribution

For the most part our work distribution has been a “free for all”, meaning the group members picked freely from what they wanted to work on during the development cycle,

within the objectives we set for that cycle. This worked out fine for the most part, but we feel we probably could have been slightly more aggressive with how many features we finished within the cycles. This left us with a bigger workload towards the end of the development as opposed to the beginning.

12.4 What could have been done differently

In hindsight, we probably could have conducted our performance testing earlier in the project, so that we could have more time to implement optimizations. We should have also had a more strict approach to what we were supposed to work on for the different development cycles, planning that out earlier as well.

12.5 Subjective views

12.5.1 Thomas Nyheim

For me, this project has had a fair bit of personal interest in its success as I am the one that is going to be using this while working for Avdeling BKI. That meant that the better this project turned out, the better for me it would be afterwards.

I feel we have created something that works well enough that it can be taken further and improved upon still. We've also conducted a small scientific test case which has been a learning experience, as it is not something often done at Bachelor levels at Gjøvik University College. There has also been a learning curve when it comes to Python and Django, which I can take with me in the future and apply to other endevours.

12.5.2 Eigil Obrestad

For me personally, this has been a very interesting project. I do think that information security is a very interesting field, and this project have been my first meeting with IDS-systems. In addition to gaining some valuable experience with software development, I have also learnt quite a bit about the IDS software Snort.

I do feel that Snowman have turned out to be a very useful tool. It is an improvement of BHTB, which was one of the main goals of the project. We provide all the functionality of BHTB in addition to several new features requested by Avdeling BKI. I personally think that the project ended up in a successfull piece of software, and as far as i can understand, I think it is going to be very helpful for the operators of the Snort-sensors at BKI.

12.5.3 Eirik Skogstad

Information security is a very interesting field, and I was very happy to be given the opportunity to develop a software for the Norwegian Armed Forces Cyber Defense. I was not much acquainted with intrusion detection prior to this project and it has taught me alot in that respect. Furthermore, the project has enabled me to become familiar with the Python programming language and the use of a web framework. In the course of the project I have also built on my experiences in software engineering and academic writing aqcuired theoretically at Gjøvik University College and practically at CERN.

Bibliography

[1] St Laurent AM. Understanding Open Source and Free Software Licensing. Sebastopol: O'Reilly Media, Inc.; 2004.

[2] Agile Software Development Process [Webpage]. New Jersey (NJ): Rapidsoft Systems; [cited 2014 May 18]. Available from: http://www.rapidsoftsystems.com/agile-development-process.html.

[3] Roesch M. SNORT - Lightweight Intrusion Detection for Networks. In: Parter DW, editor. Proceedings of the 13th Conference on Systems Administration (LISA-99), Seattle, WA, November 7-12, 1999. USENIX; 1999. p. 229–38.

[4] Snort Community Page [Webpage]. Sourcefire, Inc.; [cited 2014 May 18]. Available from: http://www.snort.org/community.

[5] Henriksen DO. Managing signatures for IDS in a distributed environment - A study of a signature management system [Master thesis]. Gjøvik University College. Gjøvik; 2012.

[6] Vaskinn C, Wikestad K. Snortmanager [Bachelor thesis]. Gjøvik University College. Gjøvik; 2012.

[7] Snort Additional Downloads Page [Webpage]. Sourcefire, Inc.; [cited 2014 May 18]. Available from: http://www.snort.org/snort-downloads/additional-downloads.

[8] Pulled_Pork [Webpage]. Google Project Hosting; [cited 2014 May 18]. Available from: https://code.google.com/p/pulledpork.

[9] Oinkmaster [Webpage]. SourceForge.net; [cited 2014 May 18]. Available from: http://oinkmaster.sourceforge.net.

[10] Graettinger CP, Garcia-Miller S, Siviy J, Syckle PJV, Schenk RJ. Using the Technology Readiness Levels Scale to Support Technology Management in the DoD's ATD/STO Environments (A Findings and Recommendations Report Conducted for Army CECOM). Pittsburg (PA): Carnegie Mellon University; 2002. CMU/SEI-2002-SR-027. Sponsored by the U.S. Army Communications Electronics Command (CECOM).

[11] Snorby [Webpage]. Snorby Community; [cited 2014 May 18]. Available from: http://snorby.org.

[12] Suricata [Webpage]. Indiana (IN): The Open Information Security Foundation.; [cited 2014 May 18]. Available from: http://suricata-ids.org.

[13] Cohen D, Lindvall M, Costa P. A State of the Art Report: Agile Software Development. Maryland (MD): Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland; 2003. DACS SOAR 11. Sponsored by Defense Technical Information Center(DTIC)/AI.

[14] Waterfall model [Webpage]. California (CA): Wikimedia Foundation, Inc.; [cited 2014 May 18]. Available from: http://en.wikipedia.org/wiki/Waterfall_model.

[15] Scrum development method [Webpage]. California (CA): Wikimedia Foundation, Inc.; [cited 2014 May 18]. Available from: http://en.wikipedia.org/wiki/Scrum_(software_development).

[16] Tornado [Webpage]; [cited 2014 May 18]. Available from: http://www.tornadoweb.org.

[17] Django v1.6 [Webpage]. Django Software Foundation; [cited 2014 May 18]. Available from: http://www.djangoproject.com.

[18] SQLite homepage [Webpage]. SQLite community; [cited 2014 May 18]. Available from: http://www.sqlite.org.

[19] Roesch M, et al. Snort Users Manual 2.9.3; 2012. Available online at http://manual.snort.org.

[20] Woodside M, Franks G, Petriu DC. The Future of Software Performance Engineering. In: Future of Software Engineering, 2007. FOSE '07. The Institute of Electrical and Electronics Engineers, Inc.; 2007. p. 171–88.

[21] Vokolos FI, Weyuker EJ. Performance Testing of Software Systems. In: WOSP '98: Proceedings of the 1st International Workshop on Software and Performance. ACM; 1998. p. 80–7.

[22] Denaro G, Polini A, Emmerich W. Early Performance Testing of Distributed Software Applications. In: WOSP '04: Proceedings of the 4th international workshop on Software and performance. ACM; 2004. p. 94–103.

[23] Myers D. On the Use of NAND Flash Memory in High-Performance Relational Databases [Master thesis]. Massachusetts Institute of Technology. Massachusetts (MA); 2008.

[24] Capturing Architectural Requirements [Webpage]. New York (NY): IBM Corporation; [cited 2014 May 18]. Available from: http://www.ibm.com/developerworks/rational/library/4706.html#N10073.

[25] Saltzer JH, Schroeder MD. The Protection of Information in Computer Systems. Communications of the ACM 17, 7. 1974;.

[26] GNU General Public Lisence v3 [Webpage]. Massachusetts (MA): The Free Software Foundation; [cited 2014 May 18]. Available from: http://www.gnu.org/licenses/gpl.html.

[27] What is Object/Relational Mapping? [Webpage]. North Carolina (NC): Red Hat, Inc.; [cited 2014 May 18]. Available from: http://hibernate.org/orm/what-is-an-orm.

[28] SQL Alchemy homepage [Webpage]. SQLAlchemy authors and contributors; [cited 2014 May 18]. Available from: http://www.sqlalchemy.org/.

[29] Python v2.7.6 documentation [Webpage]. Oregon (OR): Python Software Foundation; [cited 2014 May 18]. Available from: http://docs.python.org/2.

[30] Should I use Python 2 or Python 3 for my development activity? [Webpage]. Oregon (OR): Python Software Foundation; [cited 2014 May 18]. Available from: http://wiki.python.org/moin/Python2orPython3.

[31] Python 2.x vs 3.x use [Webpage]. Oregon (OR): Python Software Foundation; [cited 2014 May 18]. Available from: http://wiki.python.org/moin/2.x-vs-3.x-survey.

[32] Django documentation [Webpage]. Django Software Foundation; [cited 2014 May 18]. Available from: http://media.readthedocs.org/pdf/django/1.6.x/django.pdf.

[33] Goodger D, van Rossum G. PEP 257: Docstring Conventions; 2012. Available online at http://legacy.python.org/dev/peps/pep-0257.

[34] The Eclipse Project [Webpage]. Eclipse Foundation; [cited 2014 May 18]. Available from: http://www.eclipse.org.

[35] LaTeX Project [Webpage]. The LaTeX project team; [cited 2014 May 18]. Available from: http://www.latex-project.org.

[36] jQuery [Webpage]. The jQuery Foundation; [cited 2014 May 18]. Available from: http://jquery.com.

[37] Bootstrap [Webpage]. Bootstrap core team; [cited 2014 May 18]. Available from: http://getbootstrap.com.

[38] Snort Configurations Page [Webpage]. Sourcefire, Inc.; [cited 2014 May 18]. Available from: https://www.snort.org/vrt/snort-conf-configurations/.

[39] Leff A, Rayfield JT. Web-Application Development Using the Model/View/Controller Design Pattern. In: Titsworth FM, editor. Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International. The Institute of Electrical and Electronics Engineers, Inc.; 2001. p. 118–27.

[40] Eby PJ. PEP 333: Python Web Server Gateway Interface v1.0; 2003. Available online at http://legacy.python.org/dev/peps/pep-0333.

[41] Snort Advisory Page [Webpage]. Sourcefire, Inc.; [cited 2014 May 18]. Available from: http://www.snort.org/vrt/advisories.

[42] Dumbpig homepage [Webpage]. Leon Ward; [cited 2014 May 18]. Available from: http://leonward.wordpress.com/dumbpig.

Appendices

A PROJECT PLAN

SNORT Rule Manager

A complete, effective and secure rule management system for SNORT®

Project Planning Document

Thomas Nyheim

Eirik Skogstad

Eigil Obrestad

Bachelor's Thesis

Department of Computer Science and Media Technology

Gjøvik University College, Spring 2014

CONTENTS

1. OBJECTIVES AND CONSTRAINS

1.1. Background

The Norwegian Armed Forces Cyber Defense (CYFOR) unit for Computer Network Defense (CND), Avdeling BKI, is responsible for detecting and stopping cyber-attacks against the critical infrastructure and command and control systems utilized by the Norwegian military. One of the tools used to carry out this task is the open source Intrusion Detection System (IDS) SNORT® (1).

SNORT® is a leading open source IDS that is used to detect potentially unwanted or malicious network traffic based on predefined patterns described in signatures. There are both free and commercial signatures available for SNORT® and as SNORT® has been highly adopted by the security community, these signatures are continually updated and maintained.

Using SNORT® has been plagued for years by the fact that it does not have a good system for handling, updating or maintaining the signatures (henceforth called rules and rule sets) used for detection, thus making it cumbersome to utilize SNORT® for security analysts.

Research has shown that a good rule management system for SNORT® will improve the workflow and efficiency of the IDS significantly and a prototype for such a system was built in 2012 called Bring Home The Bacon (BHTB) (2). This prototype has been in use at Avdeling BKI since 2012, but is suffering from lacks in functionality and efficiency due to being a proof-of-concept prototype. The prototype did, however, prove that there is a significant gain from having an intuitive Graphical User Interface (GUI) by making the operator workflow much more efficient, as you go from having to do complex operations in a Command Line Interface (CLI) to just push buttons to make the system do things automagically (2).

There was another similar project conducted at Gjøvik University College (GUC) in 2012 called Snortmanager (3) that also deals with managing SNORT® rules. However, that project is not properly released as open source software, the project files are available but there is no documentation or instructions publically available that describes usage and installation. The project has also not been maintained since its initial release, is designed for a specific company in mind and is heavily designed around the user interface. Enabling, disabling and configuring rules is also cumbersome in this system as this is based on plaintext arguments in "policies" that users must construct themselves. The rule distribution is also done manually

from the files generated. This makes Snortmanager not very usable for Avdeling BKI and their routines.

Avdeling BKI has therefore tasked a group of students from GUC to design and develop a more complete, effective and secure rule management system for SNORT® that can continue to live on beyond this project.

1.2. Project Objectives

Results

Research, design and develop a system that as a minimum;

- Is programmatically more efficient than BHTB.
- Is developed as a standalone engine that responds to API calls from any arbitrary user interface.
- Is modular in design and allows future changes and additions to not affect existing functionality.
- Is scalable enough to handle both small and large sets of IDS sensors and rules in a production environment.
- Is able to download rules and rule sets from multiple customizable sources.
- Is able to distribute different rules to different IDS sensors.
- Can download and distribute automatically at set intervals.
- Is capable of exporting rules for sharing.
- Has the capability to activate, deactivate, threshold and suppress rules on a per sensor basis, in a way that is user friendly.
- Supports multi-user interaction.
- Can efficiently and safely work with different installations of SNORT® and various other third party tools, such as SNORBY.
- Is easy to install.
- Has an internal logging system for all activity.
- Is well documented and can be easily adopted by anyone as open source software.

In addition, the project will attempt to add the following functionality;

- A separate graphical user interface.

- Said user interface must be user friendly and intuitive, and require only limited knowledge of SNORT® rules from the user.
- The system can also distribute rules to Suricata, another SNORT®-like IDS, which can operate with the exact same rules as SNORT®.
- Implement security features that ensure confidentiality of rules.
- The project will investigate and potentially implement multi-threading technology to exploit modern hardware capacities.
- The system is capable of validating and checking rules for errors before they are distributed to production sensors.
- The system allows commenting of the rules.
- The system supports writing custom rules and editing existing rules.

Once the system has been developed, it will be released as an Open Source Software to the public in a well-documented format.

Effects

Forming baselines from the current prototype BHTB and from Snortmanager, the project has a goal to further increase the efficiency of rule management by investigating and implementing programming techniques such as multi-threading and a change-based system to reduce latency and delays in the system by as much as 50% compared to the other projects. The project will therefore be measuring for this effect.

Furthermore, the project will research and implement safeguards against unintentional failures in SNORT® as a result of user error and malformed or misconfigured rules, thus reducing the potential downtime of the IDS sensors, as a SNORT® sensor will simply not restart if a single rule is malformed or misconfigured. Efficient ways of solving this problem will have to be researched.

1.3. Constraints

Time

The project will commence on Febuary 3rd 2014 and will end on May 19th 2014, elapsing 15 weeks. It will comprise of an initial planning and design phase lasting 2 weeks, followed by a development period of 8 weeks, and then ending with production of a report for the last 5 weeks. During the project, to ensure health and motivation, no work shall be conducted on Sundays.

Personnel

The project group consists of three students. Additionally, the group disposes one mentor from GUC and one mentor/representative from Avdeling BKI.

Technology

During development and testing, the project will utilize a virtualized server to host a testbed of multiple sensors.

Economy

Economic constrains are established in the Project Agreement. There is no budget for this project and any costs will be compensated as per the Project Agreement on an ad hoc basis.

Legal

The project group may during development see a need for testing the system and SNORT® in a setting with live network traffic. This must be done in a controlled fashion and consider the need for confidentiality in personal data.

The project will be influenced by the two previous projects surrounding the same type of software and must be wary of copyright infringements. If anything is reused, credit must be given where credit is due.

SNORT® is also a registered trademark of Sourcefire, Inc, and the project must treat it as such. (4)

2. SCOPE

2.1. Subject

Software Engineering, Information Security.

2.2. Limitations

As SNORT® is primarily used in conjunction with the Linux operating system, this project will limit its development of software to this operating system only. More specifically the project will contain its development to the Ubuntu distribution as this is the platform utilized by Avdeling BKI. The project will, however, strive to make the software portable.

The project will also have its main focus on the systems functionality and efficiency from a programmatically standpoint and developing a graphical user interface (GUI) is considered a secondary task and will prioritized as such. A GUI for demonstration purposes will be included in the project and it will primarily take inspiration from BHTB.

While the system itself will not handle any data that may comprise of personal data, it may contain rules and rule sets that are considered classified or confidential. The system must therefore be built with this in mind and ensure confidentiality within the boundaries of the system. The project will not guarantee the confidentiality of the system outside these boundaries.

These boundaries include the program and its processes and threads, and the network link established between the system and remote sensors. The project assumes that the system owner has already established database security.

2.3. Project description

In broad strokes, this project is about designing a system that is a highly modular API-based engine, which is independent of any GUI. The project will be the spiritual successor to BHTB, but also take some inspiration from Snortmanager. Essentially taking good ideas from both projects, add a few of our own, and create a more adoptable open source rule management system for SNORT®.

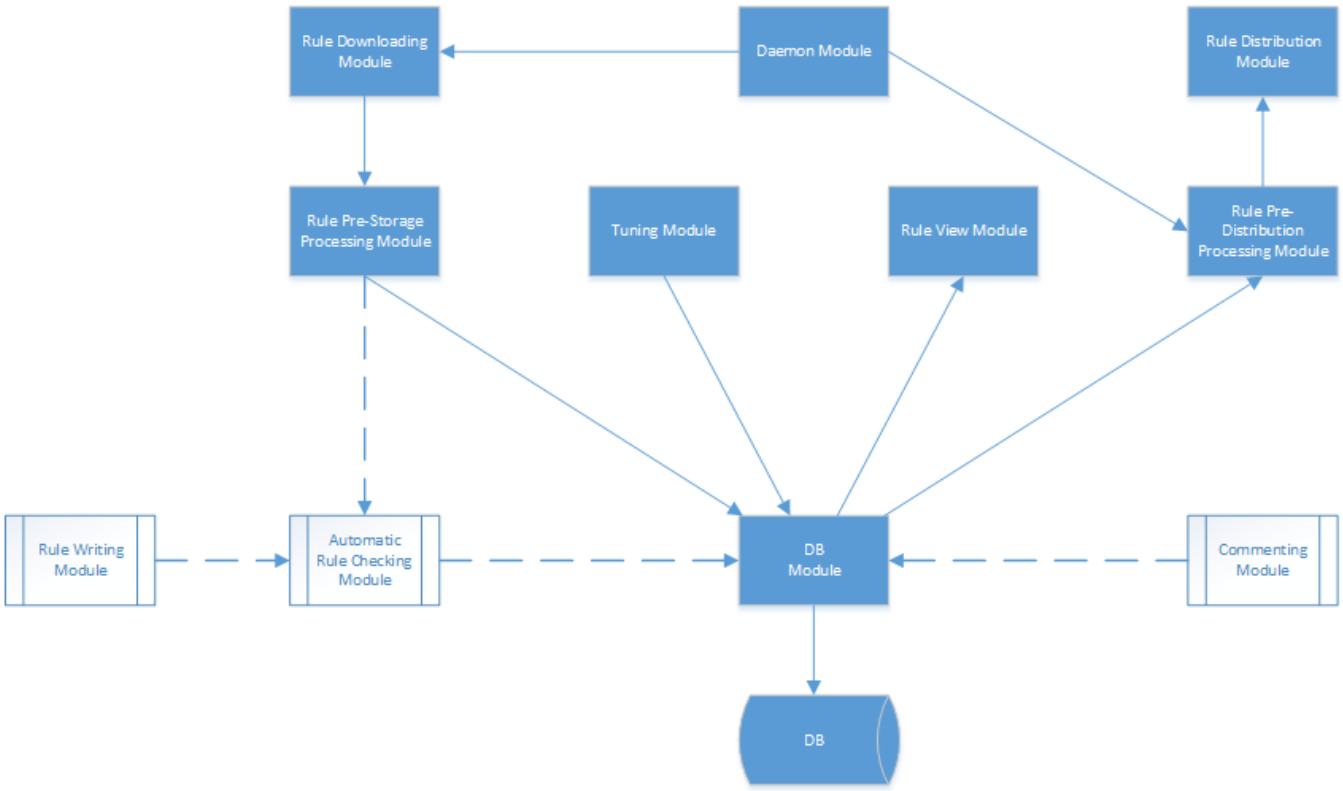
We will utilize the latest stable version of Python as the main programming language for the project. C++ was also considered, but due to the limited time at our disposal, we feel we can

create more functionality with Python, due to it requiring less memory management and general “programmer friendly” ways of doing things.

The system will require a database to function efficiently and the project will utilize MariaDB, as this is the de facto standard for the Linux platform. MariaDB is a separate branch developed from MySQL by the original developer of MySQL, which is now owned and maintained by Oracle.

Furthermore, the project will also consist of creating a GUI that is user friendly and responsive. We will base this heavily on the GUI developed for BHTB in terms of displaying rules in lists and with buttons and checkboxes to manipulate them. We may also borrow some design ideas from Snorby (5), another SNORT® tool. The main technologies we will utilize for the GUI will consist of standard web-oriented languages such as HTML, CSS, PHP and/or Javascript.

Project modules



The rule “engine” can essentially be boiled down into 11 modules, which we will base our development around. The blue modules must be in place to consider the system “functionally complete”.

Rule Downloading Module

This module will handle all functionality related to downloading rules.

Rule Pre-Storage Processing Module

This module will essentially take downloaded rules and process them so that they are ready to be fed into the database.

DB Module

This module will handle all input and output to/from the database.

Rule Pre-Distribution Processing Module

This module will essentially take rules stored in the database and prepare them for distribution.

Rule Distribution Module

This module will be in charge of distributing rules out to the SNORT® sensors.

Tuning Module

This module will handle all functionality related to turning rules on or off, thresholding them or suppressing them.

Rule View Module

This module will be similar to the Rule Pre-Distribution Processing Module, but instead of preparing the rules for distribution to SNORT® sensors, it will prepare them for a human readable format or GUI format.

Daemon Module

This module will run continually on the system to ensure that the API is responsive and to carry out any interval based tasks, such as automatic updates or distributions.

Commenting Module

This module will allow users to attach comments to rules.

Rule Writing Module

This module will handle functionality for writing custom rules and editing existing rules.

Automatic Rule Checking Module

This module will be responsible for verifying rules to ensure that there are no malformed or misconfigured rules that may potentially break a SNORT® sensor.

3. PROJECT ORGANIZATION

3.1. Roles

Project Manager

Thomas Nyheim

Development Team

Thomas Nyheim

Eirik Skogstad

Eigil Obrestad

Project Mentor

Slobodan Petrovic

Client Representative

Jarle Kittilsen

3.2. Regular meetings

The project will conduct regular status meetings every Friday. These meetings will be alternating every other week between who is participating among the project mentor and the client representative. Meaning the group will meet with the project mentor every two weeks and the client representative every two weeks.

These meetings will be timed so that the meeting with the project mentor happens in the middle of a development cycle and the meeting with the client representative will take place at the end of a development cycle. This ensures that the project group can get feedback from the mentor during the development and get feedback from the client on our progress and direction.

3.3. Rules

The rules exist to ensure an effective execution of the project and to ease the handling and resolution of potential conflict situations.

1. Group leader. The project group shall have one group leader who is elected for the entire duration of the project. The group leader is responsible for hosting weekly group meetings and keeping an overview of the project's work progress.

2. Work. Each group member is obliged to complete their work as best they can in accordance to what is agreed upon in the work-planning meetings. It is expected that each group member devotes a significant amount of time to the project and that the work load is, as far as possible, equally divided between all group members. Lack of participation from a group member will be discussed with the project mentor and repeated deliberate failures to produce the expected amount of work might ultimately lead to an exclusion from the group. Also, no group member is permitted to perform any work related to the project during Sundays unless this is approved by all group members.

3. Conflicts. Any disagreement regarding a decision shall be resolved by voting by the majority rule. Should the majority for some reason be unclear, the project leader will have the final say. If the decision in question has an important consequence for the end-product of the project, the project mentor or client representative should normally be consulted before making a decision.

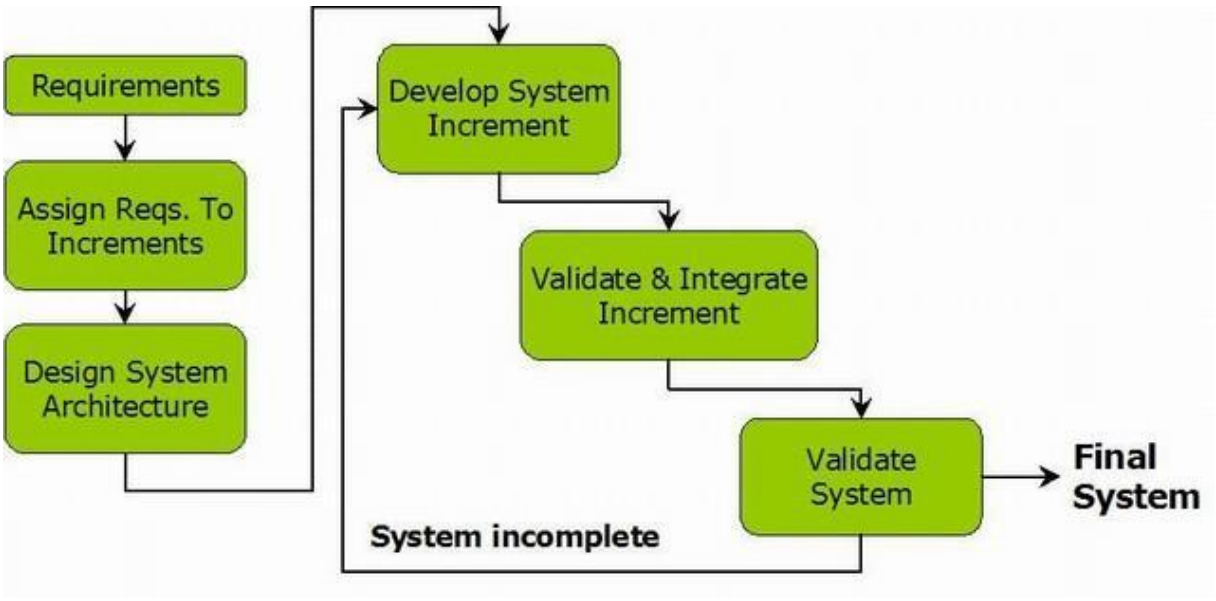
4. Expenses. Any expenses borne by a group member in direct relation to the project shall be reimbursed fully or partially by the other group members if this is decided by the majority of the project group. If the car of a group member is utilized for project-related transportation, the fuel cost shall be equally split between participating group members.

5. Absence. Group members are obliged to report to the other members any previsioned absence that directly interferes with the project work. Absence from the project lasting one consecutive week or more, or multiple unauthorized absence periods, is not permitted as this can severely endanger the work progress.

4. DEVELOPMENT MODEL

As this project is time-restricted, the development should be as efficient as possible. We therefore need a model that utilizes the manpower we have present in the best possible way. The size of the group is also quite small, so big and complex development models would add too much bureaucracy to coordinate the project and is thus overkill.

As we want to have a lightweight development-model, some form of an agile development model with well-defined development-cycles is preferred. This way, each group-member can take an active part in the project and all members would feel a sense of ownership.



Developing iterative and incrementally, we are getting the best from both the unstructured world, and from the very well defined waterfall approach. We are able to get feedback during the project, and have a structured and reasonable way of actually doing changes in the plan while the project is running. At the same time we would be able to maintain the overall plan within cycles, so that it is possible to work individually, while still being well coordinated.

To be able to develop all the modules of the project within the timeframe, each development cycle will consist of multiple modules developed in parallel. This allows us to maximize our utilization of time and personnel.

The start of each cycle will consist of a meeting where we define the goals for that cycle, and identification of the individual tasks that we need to get done. This is similar to the start of a sprint in the development model Scrum.

In the development part of the cycle, we will individually work on the tasks that were defined in the planning-part. Regular meetings and cooperative programming within the group would ensure that we all pull the project in the same direction.

At the end of each cycle we should have a working set of software, which ideally should comply with all the goals set in the start of each cycle. We are planning to have a meeting with Avdeling BKI between each cycle to present the progress, and to let them influence the direction of the project.

After all development cycles have been complete, we will then make final assurances that everything works as intended and that all modules coexist in harmony. This part of the development will also consist of hands-on testing by the client and various performance testing.

5. QUALITY ASSURANCE

5.1. Documentation

The source code will be documented as per the standards for Python programming. The project will also use a wiki-based documentation that will explain all functionality and API calls in a user friendly way. This wiki will be used as the project website.

5.2. Configuration control

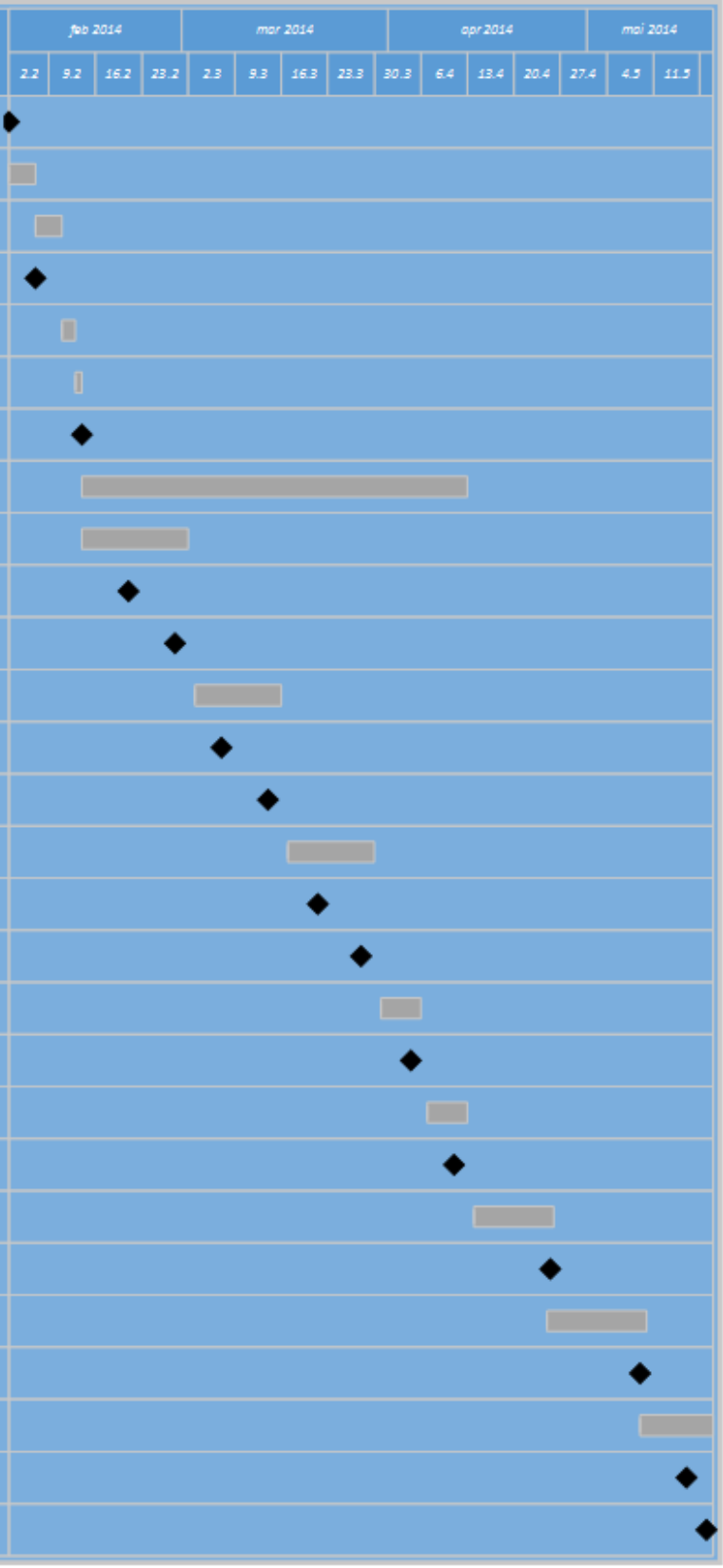
The project will utilize Subversion in a standard directory structure to ensure configuration control. The software will be versioned from 0.1 to 1.0.

5.3. Risk analysis

Threat	Likelihood	Consequence	Mitigation strategy
Project fails to complete in time	5	8	Ensure that proper planning is conducted before project start and to not graps over too much in the allotted time.
Project files or data is lost	2	10	Operate with backups routines and version control.
Sabotage of the project from an external threat agent in the form of either damage or planting of malware/backdoors	1	9	Ensure configuration control and internal project security at all times.
Project fails to meet customer expectations	3	8	Facilitate constant feedback and discussions with client to assure that the project is heading in the right directions at every step.
Loss of, or lack of participation from, project personell	3	8	Ensure availability and motivation of project personell before project start.
Illness among project personell	4	6	Maintain regulatory health and safety standards. Ensure time for breaks, food consumption and health-inducing actitivies.

6. PROJECT TIMELINE

ID	Task Name	Start	Duration
1	Project Start	03.02.2014	0d
2	Requirements Analysis	03.02.2014	4d
3	Object and Module design	07.02.2014	3d
4	Mentoring Meeting	07.02.2014	0d
5	Database design	11.02.2014	2d
6	Security analysis	13.02.2014	1d
7	Implementation Decision Meeting	14.02.2014	0d
8	GUI Development	14.02.2014	50d
9	Development Cycle 1	14.02.2014	14d
10	Mentoring Meeting	21.02.2014	0d
11	Client Meeting	28.02.2014	0d
12	Development Cycle 2	03.03.2014	12d
13	Mentoring Meeting	07.03.2014	0d
14	Client Meeting	14.03.2014	0d
15	Development Cycle 3	17.03.2014	12d
16	Mentoring Meeting	21.03.2014	0d
17	Client Meeting	28.03.2014	0d
18	Development Cycle 4	31.03.2014	6d
19	Mentoring Meeting	04.04.2014	0d
20	Final Completion Testing	07.04.2014	6d
21	Client Meeting	11.04.2014	0d
22	Report Draft 1 Writing	14.04.2014	11d
23	Report Draft 1 Review Meeting	25.04.2014	0d
24	Report Draft 2 Writing	25.04.2014	13d
25	Report Draft 2 Review Meeting	09.05.2014	0d
26	Report Final Draft Writing	09.05.2014	10d
27	Report Final Draft Review Meeting	16.05.2014	0d
28	Project Completion	19.05.2014	0d



APPENDIX A - CITATIONS

1. SourceFire Inc. About SNORT. [Online]. Available from: http://www.snort.org/snort.

2. Henriksen DO. Managing signatures for IDS in a distributed environment - A study of a signature management system. Gjøvik University College; 2012.

3. Vaskinn C, Wikestad K. Snortmanager. Gjøvik University College; 2012.

4. SourceFire Inc. Snort Trademark Guidelines. [Online]. [cited 2014 January. Available from: http://www.snort.org/legal/snort-licensing/snort-trademark-guidelines.

5. threat stack, inc. Snorby Website. [Online]. Available from: https://snorby.org/.

B CONTRACTS

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Avdeling for Beskyttelse av Kritisk Infrastruktur, Cyberforsvaret, Forsvaret (BKI) (Oppdragsgiver), og

Thomas Nyheim (Student),

Eirik Skogstad (Student) og

Eigil Obrestad (Student)

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 27. Januar 2014 til 19. Mai 2014.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG 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 HiG 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/fax, reiser og nødvendig overnatting på steder langt fra HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
   - 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. HiG 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 faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG 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 HiG og/eller studenter har interesser.

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

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 Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer 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, skjer dette uten HiG som partner.

   Eiendomsrett for resultatet av dette prosjektet er beskrevet ved egen avtale (vedlagt).

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

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): Slobodan Petrovic

Oppdragsgivers
kontaktperson (navn): Jarle Kittilsen

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): _____ dato _____

IMT Dekan/prodekan (signatur): _____ dato _____

HØGSKOLEN I GJØVIK

AVTALE OM EIERSKAP AV ÅNDSVERK

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Avdeling for Beskyttelse av Kritisk Infrastruktur, Cyberforsvaret, Forsvaret (BKI) (Oppdragsgiver), og

Thomas Nyheim (Student),

Eirik Skogstad (Student) og

Eigil Obrestad (Student)

Avtalen angir avtalepartenes eierskapsforhold til åndsverk, kildekode og programvare produsert under prosjekt med arbeidstittel «Snort Rule Manager» utført ved HiG av Nyheim, Skogstad og Obrestad for BKI:

1. Ved prosjektets sluttførelse den 19. mai 2014 vil all kildekode, programvare og dokumentasjon bli publisert som åpen kildekode i henhold til lisensen GNU General Public License[1].

3. Oppdragsgiver stiller fritt til å benytte programvaren etter de retningslinjer som er angitt i lisensen beskrevet i punkt. 1.

4. HiGs eierskap defineres i Prosjektavtale punkt 2 og 4, samt de retningslinjer som er angitt i lisensen beskrevet i punkt 1.

5. Partenes signaturer:

HiGs veileder (navn): Slobodan Petrovic

Oppdragsgivers kontaktperson (navn): Jarle Kittilsen

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): _____ dato _____

IMT Dekan/prodekan (signatur): _____ dato _____

Versjon Januar 2011hb

[1] GNU General Public License - http://www.gnu.org/licenses/gpl.html

C MEETING RECORDS

C.1 Meeting Minutes

Referat Møte 12.01.2014

Deltagere: Nyheim, Obrestad, Skogstad

- Nyheim settes til rollen som prosjektleder.
- Oppgaven skrives på Engelsk som hovedspråk.
- Gruppen sikter mot karakteren A i dette prosjektet.
- Skogstad har forelesninger tirsdag, onsdag og torsdag, Nyheim har forelesninger mandag, tirsdag og onsdag.
- Tentativ fast møtedag settes til fredag.
- Skogstad settes til å skissere et gruppereglement.
- Obrestad settes til å skissere en utviklingsmodell med utgangspunkt i en agile/iterativ modell.
- Grunnet de andre medlemmenes arbeidsbelastning hos CERN kommer Nyheim til å ta mye av arbeidet med forprosjektet, dette godtas av Nyheim uten problemer.
- Gruppen tentativt foreslår at prosjektet benytter en wiki for dokumentering av kode, må undersøke med IT-tjenesten om dette kan ordnes der.
- Gruppen avtaler at det bør settes av en dag til at alle medlemmene får besøkt BKI så tidlig som mulig etter at Obrestad og Skogstad er ferdig hos CERN ca 2. feb.

Referat Møte 15.01.2014

Deltagere: Nyheim, Obrestad, Skogstad, Petrovic

- Petrovic nevner at i tillegg til SNORT kan SNORT regler benyttes i et system kalt Surikata og prosjektet bør ta høyde for dette.
- Petrovic trekker frem Snort Manager som et annet tidligere prosjekt som også håndterer SNORT regler, selv om det prosjektet også trakk inn andre elementer rundt konfigurasjon rundt SNORT sensorer. Det anbefales derfor at prosjektet fokuserer noe på å gjøre det mer unikt fra tidligere oppgaver.
- Noen av punktene for målene bør inneholde mål som brukervennlighet, hastighet og sikkerhet. Det bør også legges inn begrensninger på områder prosjektet ikke kommer til å ta for seg.
- I en diskusjon utenfor dette møtet bestemmer gruppen seg for å benytte Python som utviklingsspråk for prosjektet.

Referat Møte 06.02.2014

Deltagere: Nyheim, Obrestad, Skogstad, Petrovic

Rapport

- Petrovic: For å få best mulig resultat bør rapporten inneholde tidsanalayser, statistikk, formler etc.
- Gruppen vil ta sikte på å lokalisere flaskehalser i BHTB, deretter forske på og planlegge løsning.
- Rapporten skrives i LaTeX. Mal finnes på nett.

Utvikling

- Petrovic: Utviklingen bør følge Løk-prinsippet med kjernen først, deretter lag for lag.
- Petrovic mente også at vi burde forkaste løsningen med klient kjørende på sensor, og heller implementere scp/ssh fra server. Vi konkluderte med at utviklingen av begge løsninger ville kreve mye av det samme forarbeidet, og førstnevnte kan brukes som en nødløsning.
- Gruppen vil ta i bruk Python og forskjellige rammeverk for å gjøre utviklingen mer effektiv.

Referat Møte 14.02.2014

Deltagere: Nyheim, Obrestad, Skogstad, Kittelsen, Heen

Agenda

- Presentasjon av design (Project Decision Meeting.pptx, arkitektur, usecaser, datamodell, m.m.)
- Innspill fra oppdragsgiver
- Drøfting av ulike filer i regeloppdateringene

Generelt

Før møtet ble det foreslått at prosjektnavnet skal være Snoman, med tilhørende snømann-logo.

Både gruppen og oppdragsgiver er enige om at det skal først og fremst fokuseres på det essensielle i utviklingen; den vitige kjernefunksjonaliteten skal være på plass før eventuell ekstra funksjonalitet, som f.eks. regel-skriving, og -kommentering samt Ole Brumm-løsninger.[1]

Innspill fra oppdragsgiver

Oppdragsgiver var generelt sett fornøyd med designet, og det ble bestemt at vi fortsetter som planlagt.

Angåande klient-løsningen ble det påpekt at dette må fungere i et langtidsperspektiv, spesielt med tanke på fremtidige systemoppdateringer. Gruppen vil ha dette i tenkene og bruke mest mulig std biblioteker. I tillegg vil programvaren pakkes for debian og eventuelt andre populære distribusjoner.

Det ble også reist spørsmål om vi kunne støtte autonome sensorer, dvs. sensorer helt isolert fra, eller uten sterk tilknytning til, serveren. Dette er i utgangspunktet utenfor oppgavens avgrensing, men gruppen satser på å lage et system der man har mulighet til å la sensoren ligge registrert på serveren uten at man kommuniserer direkte med den. Gruppen vil lage funksjonalitet for å eksportere regler/regelsett slik at disse manuelt kan sendes til autonome sensorer.

Filer i regeloppdateringene

classifications	Vi parser filen, legger klassifiseringene inn i databasen (core.RuleClass), og genererer denne filen for sensorene. Vi sletter aldri noe fra denne tabellen automatisk.
gen-msg	Som classifications, ingen sletting.

1 Ja takk, begge deler

sid-msg	Parses, og legges inn i Rule.msg
unicode	
reference	
snort_conf	Skal ikke statisk sendes til sensoren, da denne inneholder all kjerneconfig. Kan med fordel endres så lite som mulig av programmet. Include egen fil der regelsett er definert.

Preproc/so-rules

Preproc/so-rules er foreløpig ikke brukt av BKI, så de kan vi ignorere. Alternativt legger vi til generators i databasen, og ett gid-felt for reglene. Funksjonalitetsmessig mtp fremtiden er det lurt å legge det til i databasen, men det vil øke tabellstørrelsene noe.

Referat Møte 21.02.2014

Deltagere: Nyheim, Obrestad, Skogstad, Petrovic

Neste møte: 07.03.2014 kl 11:00

Agenda

- Fremgang siden sist
- Rapporttips fra Petrovic

Fremgang siden sist

Gruppen har komt godt igang med utviklingen. I første utviklingssyklus har gruppen fått på plass datamodellen, det meste av regelimportering er på plass og det har blitt jobbet mye med GUI-et. I tillegg har gruppen fått på plass grunnleggende synkronisering mellom tjener og klient.

Rapporttips fra Petrovic

Petrovic gjentar at rapporten må inneholde noe spesielt dersom den skal kunne vurderes til toppkarakter. Med dette menes konkret forskning som presenteres med målinger/matematiske modeller. Det ble foreslått at gruppen definerer noen «metrics» for ytelse o.l. som man kan bruke for å sammenligne prosjektet med tidligere prosjekter, herunder Snortmanager og spesielt BHTB.

For å starte dette arbeidet bør gruppen definere forskjellige kriterier om hva som skiller et bra og et dårlig system, og deretter måle alle kjente tidligere prosjekter opp mot dette og projisere en samlet ytelsesvurdering som vektorer i rommet eller planet. Her ble det også nevnt bestemmelse av optimal og euklid distanse for vektorene.

Gruppen har fastsatt noen formelle ytelseskrav i kravspesifikasjonen som kan tjene som en basis for sammenligning med BHTB. Det bør også fokuseres på å sammenligne prosjektet med Snortmanager så lang det lar seg gjøre, og inkludere en drøfting om denne problemstillingen.

Gruppen har også fastsatt noen sikkerhetsutfordringer som bør utredes i rapporten.

Referat Møte 28.02.2014

Deltagere: Nyheim, Kittelsen, Heen, Øvrig BKI personell

Agenda

- Presentasjon av fremdrift, hovedsakelig vise frem GUI.

Generelt

Det ble vist frem at man kan oppdatere regler gjennom GUI og det ble vist frem Rules siden. Oppdragsgiver var i all hovedsak fornøyd, dette er stort sett det samme som de har fra før.

Innspill fra oppdragsgiver

Det ble ytret ønske om å få inn en regels classification/severity i Rules listen.

Det ønskes også at bredden på GUI tilpasses bredere skjermer, på 1080p blir det for smalt.

Det spørres også om det vil bli mulig å kopiere regelsett oppsett fra en sensor til en annen på en enkel måte.

Referat Møte 14.03.2014

Deltagere: Nyheim, Kittelsen, Brein-Riise

Agenda

- Presentasjon av fremdrift, hovedsakelig vise frem GUI.

Generelt

Det ble vist frem de endringer som var fra sist. Hovedsakelig regel, regelsett og regelklasse lister. I tillegg ble tuning vist frem. Oppdragsgiver var stort sett fornøyd.

Innspill fra oppdragsgiver

Det ønskes «tooltips» når man holder over et felt slik at en enkelt kan forstå dataene i listene.

Det er en bug i hovercolor i regellista.

Det ønskes at i input-feltene til thresholding og suppress skjemaene, står tekst fra Snort manualen om datafeltet.

Det ønskes timestamping av alle aktiviteter.

Det ønskes at kilde for regler og regelsett står i listeraden.

Det ønskes at listene har en hiarkisk oppbygning, hovedsakelig i regelsett og oppdateringslistene.

Det ønskes at den rå regelstrengen gjøres mer synlig i regellista.

Referat Møte 24.03.2014

Deltagere: Nyheim, Obrestad, Skogstad, Petrovic

Neste møte: 04.04.2014 kl 10:00

Merk: gruppen vil demonstrere systemet på neste møte.

Agenda

- Fremgang siden sist
- Hva skjer fremover
- Forskning i rapport

Fremgang siden sist

Mye av systemet er ferdig. Regler inn, manipulering av regler, og regler ut på sensor forventes å være ferdig etter development cycle 3 (DC3).

Hva skjer fremover

DC4 vil hovedsaklig omfatte bugfixing og implementering av eventuell ekstra funksjonalitet. Testing-fasen vil muligens komme før DC4, dersom gruppen anser dette som fornuftig. Etter DC4 vil gruppen ha hovedfokus på rapporten. Rapportarbeidet har allerede startet, og en mal for rapporten er klar.

Forskning i rapport

Gruppen presenterte planer for testing og sammenligning av programvaren. Petrovic gjentok at det var viktig å definere klare kriterier og presentere målinger som vektorer og måle distansen mellom disse. Gruppen ønsker også å få frem forholdet mellom kompleksitet og ytelse etterhvert som kodebasen har ekspandert.

Referat Møte 01.04.2014

Deltagere: Nyheim, Obrestad, Skogstad, Kittilsen

Agenda

- Demonstrasjon av Snowman

Demonstrasjon av Snowman

Gruppen demonstrerte all funksjonalitet i Web-grensesnittet. BKI forventer å motta programvare for testing fredag 04/04/14. Det ble diskutert hva «Dashboard» siden bør inneholde. JK foreslo følgende:

- Hvilke sensorer har vi kotakt med?
- Når skjedde siste oppdatering og synkronisering
- Oversikt over nøkkeltall (eks. antall regler)

Denne siden trenger heller ikke være komplett, da BKI vil kunne ønske å fylle den med tilpasset informasjon. Det ble også vektlagt at systemet må dokumenteres grundig. Gruppen har allerde mye dokumentasjon i raportens «Implementation» kapittel som kan brukes som basis for dokumentasjonen.

Referat Møte 04.04.2014

Deltagere: Nyheim, Obrestad, Petrovic

Neste møte: 28.04.14

Agenda

- Demonstrasjon av Snowman

Demonstrasjon av Snowman

Gruppen demonstrerte all funksjonalitet i Web-grensesnittet. Til neste møte ønsker Petrovic å lese rapport.

Referat Møte 28.04.2014

Deltagere: Petrovic, Nyheim, Obrestad, Skogstad

Agenda

- Rapport utkast 1
- Rapport endelig utkast

Neste møte: 16. Mai kl 10:00.

Rapport

Gruppen leverte første utkast av rapporten for gjennomlesning. Petrovic var generelt fornøyd med oppsettet. Følgende kommentarer til innhold:

- Skriv Snort uten registrert R
- Test av update med 10 regler per fil: er 10 regler typisk? Hvorfor 10?
- Test case navn: rules per file overflødig
- Teoretisk analyse av forventet test resultat, eks. Utdyp hvorfor linær tid forventes
- Performance testing bør omdøpes til Performance analysis og deles inn i teoretisk analyse (pre-test) og praktisk analyse
- Resultattabell bør omstruktureres
- Få frem hva som er bedre med snowman ift bhtb/sm, ofres effektivitet for funksjonalitet?
- Forskjeller nevnes i introduksjon

Endelig utkast leveres til Petrovic 13. Mai.

C.2 Sprint Planning Meetings

Sprint Planning Notes 17/02/2014

kommentering logging syntax

Regler inn: (Automatisk innlasting av større sett)

- parse heile tar fil
- lese classifications.config først -> ruleClass objekt
- parse gen-msg.map -> Generator
- parse reference.conf -> ruleReferenceType
- parse rules -> Rule, RuleSet (50- parse sig-msg.map

- Update/files/..

GUI: Vise regler

Eirik koder regler inn, Thomas tar seg av GUI, Eigil begynner å se på RPC.
Kode klar til Fredag 21.02.14

Sprint Planning Notes 03/03/2014

Hva ble gjort forrige:
Regler inn:
Legge til og endre kilder.
Laste opp og parse manuelle filer.
Laste ned og parse fra kilder.
GUI:
Kan se uttømmende liste over regler.
Listen er søkbar.
Kan se detaljer for enkeltregler.
Kan legge til og endre kilder.
Kan laste opp og starte parsing av regler fra manuelle filer.
Kan starte nedlasting og parsing av regler fra kilder.
Hva henger igjen:
Tuning som allerede er i regler blir ikke parset pr nå.
Brukeren får ikke noen oversikt over endringer fra siste oppdatering i GUI, men det er implementert i databasen.
Hva skal gjøres denne sprint:
Rapport:
Begynne å skrive implementerings-kapitler for de delene av systemet en har implementert.
Få med fancy designvalg.
Skal være "ferdig" til torsdag.
Tuning-modul:
Slå av og på regler og regelsett.
Legge til og endre threshold og suppress.
GUI:
Regel sett liste
Regel klasse liste

Sensor administrasjon
Funksjonalitet for å støtte tuning.
Liste over endringer i siste regel-oppdatering.
Tilbakemeldinger til brukeren på jobbing i backend.
Tillegg:
Legge til threshold-parameter på Rule (Default-instillinger for regelen)
Legge til supress-parameter på Rule

Sprint Planning Notes 17/03/2014

(Eigil)
Lage klient
(Eirik)
Parse threshold og suppress riktig, update modul, configfil ta alltid threshold ut av strengen.
Rename threshold. Må kunne parse threshold-filer og event filter filer! Rename thresold i all kode! detection filter må i regel-objekt, vil overkjøre eventuelt event filter.
Fortsette med rapport, implementation
regler inn aktiv ikke aktiv? config!
Sette threshold og suppress per regel og per sensor
(Thomas)
Støtte nivåer med regelsett
Regelendringer siste per oppdatering med mulighet for regel av på, ferdig->delete liste (ikke fordelt)
slette gamle revisjoner (config: number of revs to keep), rull tilbake til gammel rev.
edit event/detection filter, suppress og sensor
timestamp for når man gjør endringer i systemet med mulighet for kommentering.
Comment tabell: ID, user, time, comment, type, foreign ID
sortering i regelliste: SID, dato, navn, regelsett, klasse
kommentere, skrive, endre regler
sjekking av regler (config for hvilke kilder som skal sjekkes default)
Neste:
Filter: se kun regler som er på.
Sortering i UI.

Sprint Planning Notes 17/03/2014

sprint planning
Eigil: kode ett eller annet, Design
Eirik: rette ting fra slobodan, implementation, testing,
Thomas: introduksjon, kravspek: use case diagram, domenemodell
-tabell over features i existing solutions.

D WORKLOG

Our worklog ended up being pretty long and detailed, so we counted it all up and present the numbers in Table 24.

	Nyheim	Obrestad	Skogstad
Prep, planning & meetings	108	80	95.5
Coding	163	238	105
Thesis	102	74	198
Total	391	392	398.5

Table 24: Worklog

E ADDITIONAL USE CASES

E.1 High-level use cases

Use case	**Manage sensors**
Primary actor	User
Purpose	Manage the Snort sensors that are attached to this system.
Description	The user can either create, edit or delete sensors from the system.
Preconditions	None
Postconditions	Sensors have been added, edited or removed from the system.

Use case	**Manage sensor groups**
Primary actor	User
Purpose	Create, edit or delete a sensor group.
Description	A sensor group is an entity that can contain a number of sensors. The user might wish to manipulate rules on a group basis rather than for one sensor at a time. The user can set up, edit or delete a sensor group and manage which sensors are part of the group.
Preconditions	None
Postconditions	A sensor group is added, changed or removed from the system

Use case	**Manage ruleset groups**
Primary actor	User
Purpose	Create, edit or delete a ruleset group.
Description	A ruleset group is an entity that can contain a number of ruleset. The user might wish to manipulate rulesets on a group basis rather than for one ruleset at a time. The user can set up, edit or delete a ruleset group and manage which rulesets are part of the group.
Preconditions	None
Postconditions	A ruleset group is added, changed or removed from the system

Use case	**Manage sources for rule updates**
Primary actor	User
Purpose	Manage the sources for rule updates on the system.
Description	The user can create, edit or delete sources for rule updates.
Preconditions	None
Postconditions	A source has been added, edited or removed from the system

Use case	**Verify rules as valid and functional**

Primary actor	Server
Description	Before the rules are stored in the database, they are validated by doing a test run in a local SNORT-installation to catch any errors.
Preconditions	One or more rules are ready to be stored in the database.
Postconditions	Valid rules are stored in the database, invalid rules are not stored and will produce an error-message.
Triggers	Central rule database update, Write rule

Use case	**Manage rule comments**
Primary actor	User
Purpose	Create, edit or remove comments on a rule or ruleset.
Description	User selects a rule or ruleset and either adds, edits or removes its comment. The change is stored in the database.
Preconditions	Rule must exist.
Postconditions	Rule has new comment or comment is removed from rule.

Use case	**Globally enable/disable a rule or ruleset**
Primary actor	User
Purpose	Turn a rule or ruleset on/off.

Description	User sends a command to enable/disable a specified rule or ruleset, and the system updates the database to reflect that the rule is enable/disabled.
Preconditions	The rule or ruleset must exist in the database
Postconditions	The rule or ruleset is deativated on all sensors **except** if this setting is overridden in a sensor's local RuleModifier.

E.2 Detailed use cases

Use case	**Write rule**
Primary actor	User
Purpose	Create a custom rule
Description	The user writes a rule via the web-interface. The rule is processed and stored in the central rule database.
Preconditions	None
Postconditions	The central rule database contains a new custom rule.

Basic flow:

1. The user selects a ruleset for the new rule and writes the rule.
2. The rule is processed and stored in the central rule database.

Extensions:

1. The user can specify a new ruleset which will be automatically created.

Error handling:

2. If the rule is found to be invalid it will not be added to the database, and the user can opt to make necessary changes and try again.

Use case	**Enable/disable a rule or ruleset on one or more sensors**
Primary actor	User
Purpose	Turn a rule or ruleset on/off on one or more sensors.
Description	A user wants to either enable or disable a rule or ruleset on one or more sensors. This is done by sending a command that specifies which rule or ruleset is to be affected, how it is affected and on which sensor or sensors are affected. The central database is updated with this new status and the change is pushed out to the affected sensor or sensors.
Preconditions	One or more rulesets or rules must be present in the central database. One or more sensor must be registered.
Postconditions	The changes to the rule or ruleset must be reflected on the affected sensor or sensors.
Triggers	

Basic flow

1. The user enables/disables a rule on a given sensor.
2. The central database changes the status of the rule or ruleset by manipulating the status flags.
3. The system notifies the sensor that a change has occurred.
4. The change is synced to the affected sensor.

Error handling:

4. The change may never be synced to the sensor due to errors: *There must be an error protocol that is followed in this scenario.*

Use case	**Rule tuning on sensor**
Primary actor	User
Purpose	Manipulate threshold or suppress configurations for a specific rule on one or more sensors.
Description	A user wants to either turn on or off a threshold or a suppression modifier on a rule on one or more sensors. This is done by sending a command that specifies which rule the modification will apply to, the parameters of the modification and for which sensor(s) this will apply for. The central database is then updated with this new status and the change is pushed out to the affected sensor(s).
Preconditions	The rule in question must be present in the central database and the sensor(s) must be registered.
Postconditions	The modifications are applied on the rule and sensor(s) in question.
Triggers	Set rule threshold, Set rule suppression

Basic flow

1. The central database creates a threshold/suppression for the specified rule and sensor(s).
2. The system notifies the sensor(s) that a change has occurred.
3. The change is synchronised to the affected sensor(s).

Extensions

1. User turns off a threshold/suppression on a rule on one or more sensors.

Error handling

2. The sensor(s) already contain(s) a threshold/suppress for said rule: *User is asked to overwrite or cancel operation.*
3. The change may never be synced to the sensor due to errors: *There must be an error protocol that is followed in this scenario.*

F PERFORMANCE TEST SCRIPTS

F.1 Lines Added to Snowman Code

```
start = datetime.datetime.now()
<code that performs update>
end = datetime.datetime.now()
timefile = open("/tmp/srm-update-timing.txt", "a")
timefile.write("Time: %s\n" % str(end - start))
timefile.close()
```

Listing F.1: Code lines used for timing.

F.2 Bash Script for Snowman

```
#!/bin/bash
URL="http://192.168.1.100/testing/"
FILE=("testset.1-1.tar.gz" "testset.10-1.tar.gz"
"testset.100-10.tar.gz" "testset.1000-1.tar.gz"
"testset.1000-100.tar.gz")
FILEF="testset.10000-1000.tar.gz"
FILE2=("testset.rev2.1000-100.tar.gz"
"testset.rev3.1000-100.tar.gz")
FILE3=("s.1.tar.gz" "s.2.tar.gz")

function runTest {

        echo "DROP DATABASE srm; CREATE DATABASE srm;" |
        mysql -usrm -pbah5oofa6booyeeJa2Da
        ../manage.py syncdb --noinput
        python createDemoData.py
        python ../bin/snowmand &
        PID=$!
        echo "INSERT INTO update_source
        (name, url, md5url, schedule, locked) VALUES
        ('Testing', '$URL$1', '$URL$1.md5',
        'No automatic updates', 0);"
        | mysql -usrm -pbah5oofa6booyeeJa2Da srm

        python runTimedUpdate.py 3
        python setRuleSetsToTestSensor.py
        sleep 25
        kill $PID

}

function runTest2 {

        echo "DROP DATABASE srm; CREATE DATABASE srm;" |
```

```
35          mysql -usrm -pbah5oofa6booyeeJa2Da
36          ../manage.py syncdb --noinput
37          python createDemoData.py
38          python ../bin/snowmand &
39          PID=$!
40          echo "INSERT INTO update_source
41          (name, url, md5url, schedule, locked) VALUES
42          ('Testing', '$URL$1', '$URL$1.md5',
43          'No automatic updates', 0);"
44          | mysql -usrm -pbah5oofa6booyeeJa2Da srm
45
46          python runTimedUpdate.py 3
47          python setRuleSetsToTestSensor.py
48          sleep 250
49          kill $PID
50
51  }
52
53  function runSpecialTest {
54
55          echo "DROP DATABASE srm; CREATE DATABASE srm;" |
56          mysql -usrm -pbah5oofa6booyeeJa2Da
57          ../manage.py syncdb --noinput
58          python createDemoData.py
59          python ../bin/snowmand &
60          PID=$!
61          echo "INSERT INTO update_source
62          (name, url, md5url, schedule, locked) VALUES
63          ('Testing', '$URL$1', '$URL$1.md5',
64          'No automatic updates', 0);"
65          | mysql -usrm -pbah5oofa6booyeeJa2Da srm
66
67          python runTimedUpdate.py 3
68          python setRuleSetsToTestSensor.py
69          sleep 250
70          python runTimedUpdate.py 3
71          python setRuleSetsToTestSensor.py
72          sleep 60
73
74          echo "UPDATE update_source SET url = '$URL$2',
75          md5url = '$URL$2.md5' WHERE id=3;"
76          | mysql -usrm -pbah5oofa6booyeeJa2Da srm
77
78          python runTimedUpdate.py 3
79          python setRuleSetsToTestSensor.py
80          sleep 60
81          kill $PID
82
83  }
84
85  for file in ${FILE[*]}
86  do
87          echo -e "$file" >> /tmp/srm-update-timing.txt
```

```
88          for i in {1..5}
89                  do
90                          runTest $file
91                  done
92          echo -e "\n" >> /tmp/srm-update-timing.txt
93  done
94
95  echo -e "$FILEF" >> /tmp/srm-update-timing.txt
96  for i in {1..5}
97          do
98                  runTest2 $FILEF
99          done
100 echo -e "\n" >> /tmp/srm-update-timing.txt
101
102
103 for i in {1..5}
104 do
105         runSpecialTest ${FILE2[0]} ${FILE2[1]}
106 done
107
108 for i in {1..5}
109 do
110         runSpecialTest ${FILE3[0]} ${FILE3[1]}
111 done
```

Listing F.2: Bash script used for testing.

F.3 Lines Added to BHTB Code

```
1 now = datetime.datetime.now()
2 <code that performs update>
3 stop = datetime.datetime.now()
4 time = stop - now
5 timefile = open('/tmp/timefile','a')
6 timefile.write("\nTime: "+str(time))
7 timefile.close()
```

Listing F.3: Code lines used for timing.

F.4 Bash Script for BHTB

```
1  rm -rf files_to_distribute/*
2  rm -rf tmp/*
3  rm -rf rules/*
4  rm DB.db
5  cp config.py install
6  python install/install.py
7  python update.py
8  python test.py
9  cd web/
10 python web.py &
11 PID=$!
12 cd ..
13 for i in {1..5}
```

```
14 do
15         python distribute.py
16 done
17 kill $PID
```

Listing F.4: Bash script used for testing.

G UNIT TESTS

G.1 Unit Tests for Update

Note that some of the test cases make use of external files with sample rules etc.

```
import datetime
from django.test import TestCase

from core.models import Rule, RuleRevision, Generator, RuleClass, RuleSet, Sensor, RuleReferenceType
from update.models import Source, Update
from tuning.models import DetectionFilter, EventFilter, Suppress

from update.tasks import UpdateTasks

class Test(TestCase):

  def setUp(self):
    # Create source and update objects
    try:
      source = Source.objects.get(name="Manual")
    except Source.DoesNotExist:
      source = Source.objects.create(name="Manual", schedule="00:00", url="", lastMd5="")

    self.update = Update.objects.create(time=datetime.datetime.now(), source=source)

    self.msg = "This is a sample message"
    self.filters = 'detection_filter:track by_src, count 30, seconds 60;threshold:type both, track by_dst, count 10, seconds 60;'

    self.rulestring = 'alert tcp any any -> any 21 (\
            msg:"'+self.msg+'"; \
            reference:arachnids,IDS287; reference:bugtraq,1387; reference:cve,CAN-2000-1574; \
            classtype: example-classtype; \
            priority:10; \
            '+self.filters+' \
            metadata:foo bar, ruleset community, bar 1; \
            gid:1; sid:2000000; rev:10)'

    self.raw = " ".join('alert tcp any any -> any 21 (\
            msg:"This is a sample message"; \
            reference:arachnids,IDS287; reference:bugtraq,1387; reference:cve,CAN-2000-1574; \
            classtype: example-classtype; \
            priority:10; \
```

```
38            metadata:foo bar, ruleset community, bar 1; \
39            gid:1; sid:2000000; rev:10)'.split())
40
41    self.allSensors = Sensor.objects.create(id=1, name="All
          Sensors")
42
43    try:
44      rule = Rule.objects.get(SID=2000000)
45      rule.delete()
46    except Rule.DoesNotExist:
47      pass
48
49
50  def tearDown(self):
51    pass
52
53
54  def test_updateRule(self):
55    # Insert the rule
56    self.update.updateRule(self.rulestring, "example.rules")
57
58    try:
59      # Verify that all related objects exist
60      rule = Rule.objects.get(SID=2000000, active=True, priority
          =10)
61      generator = rule.generator
62      ruleset = rule.ruleSet
63      ruleclass = rule.ruleClass
64      revision = rule.revisions.get(rev=10)
65      detectionFilter = rule.detectionFilters.get(sensor=self.
          allSensors)
66      eventFilter = rule.eventFilters.get(sensor=self.allSensors
          )
67    except Rule.DoesNotExist:
68      self.fail("Rule does not exist")
69    except Generator.DoesNotExist:
70      self.fail("Generator does not exist")
71    except RuleSet.DoesNotExist:
72      self.fail("RuleSet does not exist")
73    except RuleClass.DoesNotExist:
74      self.fail("RuleClass does not exist")
75    except RuleRevision.DoesNotExist:
76      self.fail("RuleRevision does not exist")
77    except DetectionFilter.DoesNotExist:
78      self.fail("DetectionFilter does not exist")
79    except EventFilter.DoesNotExist:
80      self.fail("EventFilter does not exist")
81
82    self.assertTrue(rule.active==True)
83    self.assertTrue(int(rule.priority)==10)
84
85    # Check revision object:
86    # 1: Check that filters are extracted
```

```
87      self.assertTrue(revision.raw==self.raw)
88      self.assertTrue(revision.msg==self.msg)
89      self.assertTrue(revision.active==True)
90      self.assertTrue(revision.filters==self.filters)
91
92      self.assertTrue(generator.GID==1)
93      self.assertTrue(ruleset.name=="community")
94      self.assertTrue(ruleclass.classtype=="example-classtype", "
            value was "+ruleclass.classtype)
95
96      self.assertTrue(detectionFilter.track==EventFilter.SOURCE)
97      self.assertTrue(detectionFilter.count==30)
98      self.assertTrue(detectionFilter.seconds==60)
99
100     self.assertTrue(eventFilter.eventFilterType==EventFilter.
            BOTH)
101     self.assertTrue(eventFilter.track==EventFilter.DESTINATION)
102     self.assertTrue(eventFilter.count==10)
103     self.assertTrue(eventFilter.seconds==60)
104
105   def test_processFolder(self):
106     UpdateTasks.processFolder(path="update", update=self.update)
107
108     # Check generator
109     generator = Generator.objects.get(GID=1, alertID=1, message=
            "snort general alert")
110     print repr(generator)
111
112     # Check reference type
113     RuleReferenceType.objects.get(name="bugtraq", urlPrefix="
            http://www.securityfocus.com/bid/")
114
115     # Get reference types generated by rule
116     rtArachnids = RuleReferenceType.objects.get(name="arachnids"
            )
117     rtUrl = RuleReferenceType.objects.get(name="url")
118
119     # Get rule and revision
120     rule = Rule.objects.get(SID=2000000)
121     ruleRevision = rule.revisions.get(rev=10, msg="DELETED
            BACKDOOR subseven 22")
122
123     # Check rule references
124     ruleRevision.references.get(reference="485",referenceType=
            rtArachnids)
125     ruleRevision.references.get(reference="www.hackfix.org/
            subseven/",referenceType=rtUrl)
126
127     # Check filter
128     rule.eventFilters.get(sensor=self.allSensors,
            eventFilterType=EventFilter.LIMIT, track=EventFilter.
            SOURCE, count=1, seconds=60)
129
```

```
130     # Check suppress
131     suppress = rule.suppress.get(sensor=self.allSensors, track=
            Suppress.DESTINATION)
132     for ip in suppress.getAddresses:
133       if ip not in ["192.168.0.1", "192.168.1.1/24"]:
134         self.fail("Suppress address not found or incorrect.")
135
136   def test_runUpdate(self):
137     UpdateTasks.runUpdate("update/test.rules")
138     Rule.objects.get(SID=2000000, active=True, priority=10)
```

G.2 Unit Tests for Web Interface

```
1  import json
2  from django.test import TestCase
3  from django.test import Client
4  from core.models import Sensor, Generator, RuleReferenceType,
       RuleSet, RuleClass, Rule
5  from update.models import Source
6  from django.contrib.auth.models import User
7  from tuning.models import EventFilter, DetectionFilter
8
9  class FilterTests(TestCase):
10   def setUp(self):
11     # Every test needs a client.
12     self.client = Client()
13
14     user = User.objects.create(username = "testuser", first_name
           = "User", last_name = "Test")
15     self.sensor = Sensor.objects.create(name="testsensor", user=
           user, active=True, ipAddress="")
16     self.sensor2 = Sensor.objects.create(name="testsensor2",
           user=user, active=True, ipAddress="")
17     generator = Generator.objects.create(GID=1, alertID=1,
           message="Generic SNORT rule")
18     ruleset = RuleSet.objects.create(name="testruleset",
           description="desc",active=True)
19     ruleclass = RuleClass.objects.create(classtype="
           testclasstype", description="desc", priority=1)
20     self.rule = Rule.objects.create(SID=2000, active=True,
           generator=generator, ruleSet=ruleset, ruleClass=
           ruleclass)
21     source = Source.objects.get_or_create(name = "Manual")
22
23   def test_AddEventFilter(self):
24     # Create an eventFilter
25     page = '/web/tuning/setFilterOnRule/'
26     data = {'comment':'no comment','force':'False','sid':'1:2000
           ', 'sensors':[self.sensor.id], 'filterType':'eventFilter
           ', 'count':'5', 'seconds':'5', 'type':'1', 'track':'1'}
27     print "Sending request for new EventFilter"
28     self.sendRequest(page, data, "filterAdded")
29
```

```
30      try:
31        f = EventFilter.objects.get(rule=self.rule, sensor=self.
              sensor)
32        f.delete()
33      except EventFilter.DoesNotExist:
34        self.fail("EventFilter was NOT created.")
35
36    def test_AddDetectionFilter(self):
37      # Create an detectionFilter
38      page = '/web/tuning/setFilterOnRule/'
39      data = {'comment':'no comment','force':'False','sid':'1:2000
            ', 'sensors':[self.sensor.id], 'filterType':'
           detectionFilter', 'count':'5', 'seconds':'5', 'type':'1'
            , 'track':'1'}
40      print "Sending request for new DetectionFilter"
41      self.sendRequest(page, data, "filterAdded")
42
43      try:
44        f = DetectionFilter.objects.get(rule=self.rule, sensor=
              self.sensor)
45        f.delete()
46      except DetectionFilter.DoesNotExist:
47        self.fail("DetectionFilter was NOT created.")
48
49    def sendRequest(self, page, data, reply):
50      response = self.client.post(page, data)
51      responseText = json.loads(response.content)[0]["response"]
52      print "Got response: "+responseText
53      self.assertTrue(responseText == reply)
```

H TEST RESULTS

Test results are split into five columns representing the actual test results, SxT1 through SxT5, followed by the mean value and standard deviation. S1, S2, and S3 represent the test hardware.

ID	System	S1T1	S1T2	S1T3	S1T4	S1T5	S1T	*S1T SD*
SYN-1	Snortmanager	0,206792	0,233198	0,210922	0,256989	0,235184	**0,228617**	*0,020362*
SYN-10	Snortmanager	0,217664	0,238152	0,388388	0,246780	0,235390	**0,265275**	*0,069632*
SYN-100	Snortmanager	1,483021	1,181549	1,264857	1,126235	1,200714	**1,251275**	*0,138694*
SYN-1000	Snortmanager	9,424450	10,498779	8,694990	9,085834	9,648248	**9,470460**	*0,678317*
SYN-1000-B	Snortmanager	1,797127	1,865735	1,850821	1,819779	1,848517	**1,836396**	*0,027541*
SYN-10000	Snortmanager	93,141342	93,759295	91,267864	92,260868	91,713406	**92,428555**	*1,020502*
SYN-1000-C	Snortmanager	1,731733	1,744372	1,724406	1,840847	1,776041	**1,763480**	*0,047548*
SYN-1000-D	Snortmanager	65,182876	64,240613	63,331489	62,854617	64,890406	**64,100000**	*0,994827*
SF-28064-A	Snortmanager	44,187545	43,474986	43,054751	43,279595	42,961623	**43,391700**	*0,487692*
SF-28064-B	Snortmanager	241,120667	247,782108	232,012255	241,237407	237,242046	**239,878897**	*5,802700*
SYN-1	BHTB	0,128643	0,129190	0,135711	0,104435	0,128201	**0,125236**	*0,012025*
SYN-10	BHTB	0,138525	0,150051	0,156831	0,155302	0,140905	**0,148323**	*0,008293*
SYN-100	BHTB	0,168435	0,171083	0,212938	0,172699	0,204697	**0,185970**	*0,021114*
SYN-1000	BHTB	0,559695	0,594925	0,543682	0,568407	0,574957	**0,568333**	*0,018930*
SYN-1000-B	BHTB	0,428708	0,443431	0,449333	0,431507	0,474401	**0,445476**	*0,018248*
SYN-10000	BHTB	3,902603	3,492913	3,591116	3,368983	3,674602	**3,606043**	*0,201163*
SYN-1000-C	BHTB	0,004394	0,004555	0,007230	0,011460	0,011442	**0,007816**	*0,003504*
SYN-1000-D	BHTB	0,553685	0,607420	0,567166	0,560094	0,530044	**0,563682**	*0,028149*
SF-28064-A	BHTB	6,018727	5,485870	5,564759	5,559401	5,563365	**5,638424**	*0,215179*
SF-28064-B	BHTB	6,068790	5,810159	4,940879	5,345223	5,867540	**5,606518**	*0,456795*
SYN-1	Snowman 0.4	1,005548	1,059578	1,080994	1,088118	1,056659	**1,058179**	*0,032370*
SYN-10	Snowman 0.4	2,332549	2,343259	2,266561	2,306506	2,198717	**2,289518**	*0,058737*
SYN-100	Snowman 0.4	17,914653	17,948775	16,687145	18,105066	16,941285	**17,519385**	*0,653922*
SYN-1000	Snowman 0.4	165,782173	166,326524	165,626654	165,380375	175,110611	**167,645267**	*4,187663*
SYN-1000-B	Snowman 0.4	146,741187	147,913234	148,554440	146,509770	146,132794	**147,170285**	*1,020374*
SYN-10000	Snowman 0.4	1898,370552	1769,044708	1768,219325	1767,793228	1772,447828	**1795,175128**	*57,717000*
SYN-1000-C	Snowman 0.4	0,312047	0,256963	0,315896	0,317985	0,278471	**0,296272**	*0,027237*
SYN-1000-D	Snowman 0.4	1,408959	1,357421	1,365136	1,387047	1,340046	**1,371722**	*0,026799*
SF-28064-A	Snowman 0.4	9084,530000	9151,330000	9235,650000	9144,080000	9263,840000	**9175,886000**	*72,918069*
SF-28064-B	Snowman 0.4	28,453345	29,113654	27,555421	28,432262	27,450084	**28,200953**	*0,694756*
SYN-1	Snowman 0.5	1.416252	1.422312	1.451006	1.377903	1.409945	**7,077418**	0,026230
SYN-10	Snowman 0.5	1.454169	1.472612	1.384907	1.469496	1.357374	**7,138558**	0,053013
SYN-100	Snowman 0.5	1.489865	1.620058	1.626211	1.556013	1.548973	**7,841120**	0,056367
SYN-1000	Snowman 0.5	2.538760	2.577367	2.571131	2.571857	2.582060	**12,841175**	0,017065
SYN-1000-B	Snowman 0.5	2.361429	2.643134	2.646817	2.445625	2.445646	**12,542651**	0,129220
SYN-10000	Snowman 0.5	14.463401	14.060287	14.196808	14.156359	14.018620	**70,895475**	0,174304
SYN-1000-C	Snowman 0.5	0.320461	0.310278	0.318357	0.280304	0.318478	**1,547878**	0,016824
SYN-1000-D	Snowman 0.5	1.448637	1.438360	1.437713	1.437401	1.395071	**7,157182**	0,020864
SF-28064-A	Snowman 0.5	60.418551	64.442370	64.315231	65.438771	65.192802	**319,807725**	2,037602
SF-28064-B	Snowman 0.5	8.937439	9.270010	9.490084	9.346018	9.154797	**46,198348**	0,208257

ID	System	S2T1	S2T2	S2T3	S2T4	S2T5	S2T	*S2T SD*
SYN-1	Snortmanager	0,098268	0,101233	0,089378	0,093957	0,084180	**0,093403**	*0,006829*
SYN-10	Snortmanager	0,097503	0,102940	0,120166	0,114301	0,102862	**0,107554**	*0,009340*
SYN-100	Snortmanager	0,129635	0,132515	0,168781	0,136476	0,127619	**0,139005**	*0,016975*
SYN-1000	Snortmanager	0,351296	0,402450	0,553614	0,684969	0,465230	**0,491512**	*0,131876*
SYN-1000-B	Snortmanager	0,385234	0,530236	0,488488	0,598950	0,543306	**0,509243**	*0,079777*
SYN-10000	Snortmanager	4,855118	5,206928	5,060140	5,503203	4,900403	**5,105158**	*0,262330*
SYN-1000-C	Snortmanager	0,818693	0,867879	0,852995	0,807754	0,798828	**0,829230**	*0,029812*
SYN-1000-D	Snortmanager	18,138424	17,636141	15,887605	15,170345	15,754937	**16,517490**	*1,291496*
SF-28064-A	Snortmanager	20,441905	20,345444	19,376373	21,248741	20,457761	**20,374045**	*0,665747*
SF-28064-B	Snortmanager	80,024188	80,004083	80,072613	79,421550	80,522949	**80,009077**	*0,391645*

SYN-1	BHTB	0,069105	0,061350	0,070685	0,065123	0,060140	**0,065281**	*0,004630*
SYN-10	BHTB	0,077151	0,069491	0,076584	0,068809	0,068088	**0,072025**	*0,004453*
SYN-100	BHTB	0,089563	0,084068	0,081976	0,082176	0,086781	**0,084913**	*0,003238*
SYN-1000	BHTB	0,147345	0,161374	0,144297	0,151398	0,148275	**0,150538**	*0,006566*
SYN-1000-B	BHTB	0,170759	0,165450	0,149698	0,243768	0,149350	**0,175805**	*0,039157*
SYN-10000	BHTB	0,883050	0,886189	0,893544	1,010669	0,877408	**0,910172**	*0,056481*
SYN-1000-C	BHTB	0,004482	0,004370	0,004460	0,004859	0,004544	**0,004543**	*0,000187*
SYN-1000-D	BHTB	0,083898	0,087971	0,087722	0,078001	0,088855	**0,085289**	*0,004497*
SF-28064-A	BHTB	1,667032	1,731853	1,727817	1,696671	1,754234	**1,715521**	*0,033997*
SF-28064-B	BHTB	2,342256	2,363745	2,493566	2,446536	2,337797	**2,396780**	*0,069617*
SYN-1	Snowman 0.4	0,701108	0,636084	0,555317	0,611619	0,609072	**0,622640**	*0,052858*
SYN-10	Snowman 0.4	1,669105	1,664763	1,665243	1,654165	1,687434	**1,668142**	*0,012127*
SYN-100	Snowman 0.4	11,571188	12,498288	12,408186	12,344324	12,163973	**12,197192**	*0,370746*
SYN-1000	Snowman 0.4	114,968508	115,748754	114,048435	113,985480	114,336628	**114,617561**	*0,742482*
SYN-1000-B	Snowman 0.4	114,456302	114,638746	113,148216	113,563002	113,701123	**113,901478**	*0,627197*
SYN-10000	Snowman 0.4	1161,359506	1162,043099	1161,683608	1164,914244	1162,343719	**1162,468835**	*1,416344*
SYN-1000-C	Snowman 0.4	0,227747	0,159089	0,159433	0,225957	0,158850	**0,186215**	*0,037102*
SYN-1000-D	Snowman 0.4	9,123599	10,065509	9,853473	9,976422	9,394814	**9,682763**	*0,405417*
SF-28064-A	Snowman 0.4	3791,930059	3821,693653	3838,703577	3818,908736	3816,684324	**3817,584070**	*16,760182*
SF-28064-B	Snowman 0.4	22,793565	21,801458	22,114177	21,775254	24,881671	**22,673225**	*1,301017*
SYN-1	Snowman 0.5	0,828667	0,809376	0,839335	0,742288	0,803740	**0,804681**	*0,037723*
SYN-10	Snowman 0.5	0,730362	0,876253	0,828412	0,778592	0,827469	**0,808218**	*0,055557*
SYN-100	Snowman 0.5	0,882882	0,773482	0,883142	0,949145	0,838829	**0,865496**	*0,064785*
SYN-1000	Snowman 0.5	1,170738	1,069663	1,024820	1,035318	1,190257	**1,098159**	*0,077281*
SYN-1000-B	Snowman 0.5	1,124399	1,191150	1,298163	1,226452	1,389368	**1,245906**	*0,101800*
SYN-10000	Snowman 0.5	5,919835	6,226214	5,894973	5,363032	5,405001	**5,761811**	*0,369023*
SYN-1000-C	Snowman 0.5	0,165332	0,161144	0,168788	0,173141	0,178974	**0,169476**	*0,006905*
SYN-1000-D	Snowman 0.5	0,764924	0,741704	0,747993	0,736324	0,750344	**0,748258**	*0,010811*
SF-28064-A	Snowman 0.5	27,017147	26,531894	26,488149	27,028858	27,310182	**26,875246**	*0,353778*
SF-28064-B	Snowman 0.5	5,109430	5,475716	5,680058	5,550670	5,629171	**5,489009**	*0,225945*

ID	**System**	**S2T6**	**S2T7**	**S2T8**	**S2T9**	**S2T10**	**S2T**	***S2T SD***
SYN-1000	Snortmanager	0,384479	0,457319	0,585513	0,490393	0,384264	**0,460394**	*0,083850*
SYN-1000-B	Snortmanager	0,399573	0,518086	0,435683	0,662156	0,465522	**0,496204**	*0,102421*
SYN-1000-D	Snortmanager	31,588423	32,522143	31,846631	31,785514	32,255841	**31,999710**	*0,379806*
SF-28064-A	Snortmanager	21,588472	20,415596	21,451682	21,566694	20,616697	**21,127828**	*0,565289*
SYN-1000	BHTB	0,167639	0,162784	0,152462	0,154713	0,154155	**0,158351**	*0,006547*
SYN-1000-B	BHTB	0,152755	0,164592	0,154168	0,154763	0,162806	**0,157817**	*0,005456*

ID	**System**	**S3T1**	**S3T2**	**S3T3**	**S3T4**	**S3T5**	**S3T**	***S3T SD***
SYN-1	Snortmanager	0,033428	0,048519	0,031483	0,040018	0,031352	**0,036960**	*0,007363*
SYN-10	Snortmanager	0,040184	0,041227	0,036264	0,035786	0,034382	**0,037569**	*0,002969*
SYN-100	Snortmanager	0,149768	0,154191	0,161973	0,162140	0,167491	**0,159113**	*0,007055*
SYN-1000	Snortmanager	1,357576	1,369004	1,358835	1,351686	1,364279	**1,360276**	*0,006621*
SYN-1000-B	Snortmanager	0,229040	0,225038	0,239134	0,219061	0,227949	**0,228044**	*0,007310*
SYN-10000	Snortmanager	13,516335	13,446996	13,514860	13,638274	13,502969	**13,523887**	*0,069951*
SYN-1000-C	Snortmanager	0,208155	0,203170	0,205232	0,202749	0,210476	**0,205956**	*0,003310*
SYN-1000-D	Snortmanager	10,703261	10,692305	10,747949	10,739760	10,700169	**10,716689**	*0,025285*
SF-28064-A	Snortmanager	8,068972	8,023264	7,940072	8,083689	8,079231	**8,039046**	*0,060308*
SF-28064-B	Snortmanager	65,141035	63,177657	64,769756	62,650342	62,393958	**63,626550**	*1,252433*

SYN-1	BHTB	0,014454	0,022309	0,022516	0,015686	0,015910	**0,018175**	*0,003908*
SYN-10	BHTB	0,021625	0,014935	0,022947	0,017066	0,021982	**0,019711**	*0,003504*
SYN-100	BHTB	0,020909	0,025678	0,031950	0,026151	0,020433	**0,025024**	*0,004682*
SYN-1000	BHTB	0,064116	0,057157	0,057794	0,059321	0,063179	**0,060313**	*0,003161*
SYN-1000-B	BHTB	0,046655	0,052058	0,052843	0,045831	0,052154	**0,049908**	*0,003372*
SYN-10000	BHTB	0,442512	0,428564	0,432250	0,447828	0,450516	**0,440334**	*0,009598*
SYN-1000-C	BHTB	0,001282	0,001301	0,001313	0,001344	0,001309	**0,001310**	*0,000023*
SYN-1000-D	BHTB	0,076648	0,088692	0,081179	0,083106	0,086751	**0,083275**	*0,004737*
SF-28064-A	BHTB	1,019142	1,008337	1,008999	1,023319	1,028006	**1,017561**	*0,008705*
SF-28064-B	BHTB	0,997865	0,996060	1,004496	1,007504	1,016197	**1,004424**	*0,008079*
SYN-1	Snowman 0.4	0,145925	0,162781	0,142633	0,142318	0,166075	**0,151946**	*0,011540*
SYN-10	Snowman 0.4	0,309486	0,328613	0,320812	0,345235	0,334128	**0,327655**	*0,013498*
SYN-100	Snowman 0.4	2,341000	2,396807	2,361303	2,326400	2,357539	**2,356610**	*0,026434*
SYN-1000	Snowman 0.4	22,487913	22,973747	23,148201	22,622529	22,919402	**22,830358**	*0,269247*
SYN-1000-B	Snowman 0.4	20,403846	20,563843	20,159649	20,159649	20,552203	**20,367838**	*0,200244*
SYN-10000	Snowman 0.4	228,432106	228,438271	226,195244	228,036478	229,088230	**228,038066**	*1,097072*
SYN-1000-C	Snowman 0.4	0,061277	0,059677	0,048918	0,055105	0,055781	**0,056152**	*0,004802*
SYN-1000-D	Snowman 0.4	16,511663	17,230070	16,638370	17,244273	17,367003	**16,998276**	*0,392596*
SF-28064-A	Snowman 0.4	1272,953816	1265,591761	1267,916730	1265,392845	1265,564088	**1267,483848**	*3,230546*
SF-28064-B	Snowman 0.4	5,235618	5,236689	5,268861	5,250577	5,311020	**5,260553**	*0,031252*
SYN-1	Snowman 0.5	0,192482	0,193482	0,223332	0,194989	0,176516	**0,196160**	*0,016928*
SYN-10	Snowman 0.5	0,194755	0,197788	0,187074	0,209426	0,186751	**0,195159**	*0,009310*
SYN-100	Snowman 0.5	0,362321	0,328652	0,335591	0,354945	0,333655	**0,343033**	*0,014698*
SYN-1000	Snowman 0.5	1,706187	1,834414	1,805236	1,972884	1,800205	**1,823785**	*0,096272*
SYN-1000-B	Snowman 0.5	0,368017	0,402124	0,487347	0,380095	0,382527	**0,404022**	*0,048160*
SYN-10000	Snowman 0.5	16,852623	16,773081	16,867794	16,782606	16,748645	**16,804950**	*0,052219*
SYN-1000-C	Snowman 0.5	0,056816	0,051502	0,054920	0,047960	0,062924	**0,054824**	*0,005651*
SYN-1000-D	Snowman 0.5	2,033637	2,039553	1,986151	2,046115	2,008360	**2,022763**	*0,024967*
SF-28064-A	Snowman 0.5	12,450983	12,513210	12,546008	12,591141	12,512589	**12,522786**	*0,051388*
SF-28064-B	Snowman 0.5	1,764425	1,681213	1,678459	1,723910	1,689931	**1,707588**	*0,036581*

Interface_Results

ID	System	S1T1	S1T2	S1T3	S1T4	S1T5	S1T	*S1T SD*
SYN-LIST-100	Snowman 0.5	2460,000	2420,000	2440,000	2690,000	2530,000	2508,000	109,864
SYN_LIST_SET	Snowman 0.5	10420	13090	11180	11310	10530	11306,000	1070,715
SYN-LIST-100	BHTB	920,120	886,855	945,417	949,077	921,335	924,561	24,939
SYN_LIST_SET	BHTB	42780,000	38570,000	39530,000	38410,000	37870,000	39432,000	1965,075

ID	System	S3T1	S3T2	S3T3	S3T4	S3T5	S3T	*S3T SD*
SYN-LIST-100	Snowman 0.5	1090,000	769,752	1130,000	1120,000	1120,000	1045,950	155,127
SYN_LIST_SET	Snowman 0.5	4690,000	4640,000	4760,000	4650,000	4640,000	4676,000	51,284
SYN-LIST-100	BHTB	639,147	683,821	643,079	674,523	646,470	657,408	20,304
SYN_LIST_SET	BHTB	15630,000	9270,000	9210,000	9220,000	9290,000	10524,000	2854,537

Sync_Results

ID	System	S1T1	S1T2	S1T3	S1T4	S1T5	S1T	*S1T SD*
SYN-1	BHTB	1,601997	1,464528	1,501998	1,470546	1,468254	**1,501465**	*0,058160*
SYN-10	BHTB	1,548066	1,465181	1,506451	1,474272	1,452849	**1,489364**	*0,038353*
SYN-100	BHTB	1,539562	1,475369	1,481972	1,486051	1,465269	**1,489645**	*0,028989*
SYN-1000	BHTB	1,863378	1,733953	1,749688	1,794258	1,713741	**1,771004**	*0,059540*
SYN-1000-B	BHTB	1,568222	1,509846	1,510405	1,513819	1,511305	**1,522719**	*0,025482*
SYN-10000	BHTB	1,834562	1,842366	1,836491	1,841225	1,844563	**1,839841**	*0,004174*
SYN-1000-C	BHTB	1,758841	1,815413	1,736488	1,766125	1,800248	**1,775423**	*0,031988*
SYN-1000-D	BHTB	1,733251	1,784421	1,816654	1,759442	1,784552	**1,775664**	*0,031216*
SF-28064-A	BHTB	1,751222	1,742123	1,782281	1,744411	1,771252	**1,758258**	*0,017660*
SF-28064-A	BHTB	2,513151	2,501121	2,496631	2,499329	2,511295	**2,504305**	*0,007431*
SF-28064-B	BHTB	1,725441	1,736654	1,782144	1,775421	1,753346	**1,754601**	*0,024321*
SYN-1	Snowman 0.5	1,584408	1,671359	1,686522	1,655096	1,630811	**1,645639**	*0,039973*
SYN-10	Snowman 0.5	1,842637	1,804390	1,867744	1,859858	1,769264	**1,828779**	*0,041271*
SYN-100	Snowman 0.5	4,091391	3,928236	3,947004	4,020025	3,890382	**3,975408**	*0,080163*
SYN-1000	Snowman 0.5	25,011648	24,825197	25,008797	24,874929	24,885310	**24,921176**	*0,084409*
SYN-1000-B	Snowman 0.5	24,842518	24,840451	25,033559	24,833728	25,034563	**24,916964**	*0,106945*
SYN-10000	Snowman 0.5	240,208419	239,224603	239,657150	240,000712	239,118943	**239,641965**	*0,473694*
SYN-1000-C	Snowman 0.5	4,237017	4,393063	4,230834	4,234789	4,233953	**4,265931**	*0,071103*
SYN-1000-D	Snowman 0.5	6,494697	6,712227	6,487460	6,538826	6,383489	**6,523340**	*0,119984*
SF-28064-A	Snowman 0.5	113,041269	112,636746	112,033961	112,276049	112,559461	**112,509497**	*0,381478*
SF-28064-A	Snowman 0.5	24,176332	24,093305	24,204543	24,276337	24,210317	**24,192167**	*0,066317*
SF-28064-B	Snowman 0.5	26,182206	26,497129	26,241492	26,299322	26,594323	**26,362894**	*0,175361*

ID	System	S2T1	S2T2	S2T3	S2T4	S2T5	S2T	*S2T SD*
SYN-1	BHTB	1,457105	1,459435	1,479865	1,468867	1,472788	**1,467612**	*0,009431*
SYN-10	BHTB	1,447054	1,466315	1,464936	1,534225	1,451509	**1,472808**	*0,035331*
SYN-100	BHTB	1.465433	1.475585	1.466679	1.477495	1.481584	**1,473355**	*0,007021*
SYN-1000	BHTB	1.515109	1.486776	1.533385	1.489662	1.492625	**1,503511**	*0,020108*
SYN-1000-B	BHTB	1,468936	1,487726	1,480308	1,480513	1,489574	**1,481411**	*0,008127*
SYN-10000	BHTB	1,617591	1,609389	1,570813	1,585638	1,586016	**1,593889**	*0,019135*
SYN-1000-C	BHTB	1,523412	1,515633	1,509423	1,516314	1,511136	**1,515184**	*0,005449*
SYN-1000-D	BHTB	1,519234	1,521990	1,513501	1,520119	1,494876	**1,513944**	*0,011120*
SF-28064-A	BHTB	1,572103	1,602751	1,587868	1,578691	1,583445	**1,584972**	*0,011531*
SF-28064-A	BHTB	2,348232	2,450012	2,412287	2,384758	2,421534	**2,403365**	*0,038638*
SF-28064-B	BHTB	1,587952	1,586874	1,603754	1,568568	1,576334	**1,584696**	*0,013310*
SYN-1	Snowman 0.5	1,500620	1,561754	1,485359	1,468405	1,502939	**1,503815**	*0,035222*
SYN-10	Snowman 0.5	1,803002	1,717718	1,694227	1,754551	1,795272	**1,752954**	*0,047404*
SYN-100	Snowman 0.5	4,080263	4,002436	4,052210	4,036057	4,009683	**4,036130**	*0,031785*
SYN-1000	Snowman 0.5	25.293685	26.148333	26.126963	24.002027	25.300082	**25,374218**	*0,874753*
SYN-1000-B	Snowman 0.5	24,543550	25,322921	26,499748	25,183168	24,591195	**25,228116**	*0,790942*
SYN-10000	Snowman 0.5	253,368507	254,918622	250,352696	250,282503	249,113682	**251,607202**	*2,429709*
SYN-1000-C	Snowman 0.5	4,393080	4,393063	4,230834	4,234789	4,233953	**4,297144**	*0,087582*
SYN-1000-D	Snowman 0.5	6,600918	6,712227	6,487460	6,538826	6,383489	**6,544584**	*0,123010*
SF-28064-A	Snowman 0.5	107,244311	108,897443	103,478098	103,113980	105,044251	**105,555617**	*2,477835*
SF-28064-A	Snowman 0.5	16,759410	16,423130	16,440110	16,392249	16,480373	**16,499054**	*0,148980*
SF-28064-B	Snowman 0.5	22,896957	23,318300	23,643247	22,009474	23,298042	**23,033204**	*0,630507*

ID	System	S3T1	S3T2	S3T3	S3T4	S3T5	S3T	*S3T SD*
SYN-1	BHTB	0,003280	0,003456	0,003255	0,003281	0,003228	**0,003300**	*0,000090*
SYN-10	BHTB	0,003492	0,003257	0,003616	0,003359	0,003587	**0,003462**	*0,000152*
SYN-100	BHTB	0,004736	0,004383	0,004759	0,004255	0,004327	**0,004492**	*0,000238*
SYN-1000	BHTB	0,014926	0,015331	0,015118	0,015460	0,014827	**0,015132**	*0,000266*
SYN-1000-B	BHTB	0,004063	0,004286	0,004142	0,004037	0,004253	**0,004156**	*0,000111*
SYN-10000	BHTB	0,120476	0,122886	0,121392	0,122107	0,121522	**0,121677**	*0,000894*
SYN-1000-C	BHTB	1,105447	1,086171	1,121011	1,103712	1,072213	**1,097711**	*0,018855*
SYN-1000-D	BHTB	1,100292	1,108830	1,028967	1,046725	1,039907	**1,064944**	*0,036839*
SF-28064-A	BHTB	1,051726	1,058626	1,072986	1,069411	1,061279	**1,062806**	*0,008512*
SF-28064-A	BHTB	1,145471	1,131565	1,119483	1,134853	1,155174	**1,137309**	*0,013625*
SF-28064-B	BHTB	1,123244	1,102139	1,109704	1,151463	1,101915	**1,117693**	*0,020772*
SYN-1	Snowman 0.5	0,359170	0,368295	0,356992	0,386964	0,386265	**0,371537**	*0,014404*
SYN-10	Snowman 0.5	0,425734	0,455836	0,468959	0,471120	0,445163	**0,453362**	*0,018677*
SYN-100	Snowman 0.5	1,147232	1,093158	1,081029	1,060226	1,091920	**1,094713**	*0,032191*
SYN-1000	Snowman 0.5	7,696620	7,705567	7,817169	7,832752	7,745720	**7,759566**	*0,062738*
SYN-1000-B	Snowman 0.5	7,319058	7,209191	7,298548	7,209002	7,224510	**7,252062**	*0,052680*
SYN-10000	Snowman 0.5	17,028875	16,950263	16,850382	16,945703	16,535579	**16,862160**	*0,193216*
SYN-1000-C	Snowman 0.5	2,154336	2,429049	2,497035	2,163776	2,149603	**2,278760**	*0,170011*
SYN-1000-D	Snowman 0.5	11,218336	11,941164	12,666270	11,209971	11,174488	**11,642046**	*0,656379*
SF-28064-A	Snowman 0.5	40,120780	40,455339	40,273142	39,852550	39,319278	**40,004218**	*0,442030*
SF-28064-A	Snowman 0.5	8,708399	8,945423	8,449702	8,699022	8,486952	**8,657900**	*0,199675*
SF-28064-B	Snowman 0.5	8,732808	9,548194	8,749769	8,627621	8,541429	**8,839964**	*0,404790*

I AVDELING BKI FEEDBACK

- Tooltip?
- Tungvindt med skrollbar på references (jeg la ikke merke til at den var der, så trodde først at noen av reference ikke ble automatisk en link)
- Sortering på kolonner
- Mulighet til å se flere enn 20 regler per side
- Status ikoner, når D (?), så kommer de over på 2 linjer
- Mulighet for å se på kun egen tuning (filtrere bort regler som kommer med tuning fra leverandør)
- Skriveleif på Emerging Threaths, skal være Emerging Threats
- Feiling av automatisk nedlastning av regler, hvordan får man med seg at dette skjer (hvis dashboard ikke blir ferdig)
- Mulighet for å søke på strenger i regelsettet?
- Mulighet for å søke etter annet enn sid og name
- Comment fungerer ikke..
- Mulighet til å hoppe til f.eks. side 1000 av 2385
- Reorganize Rules?
- Mye scrolling når man ser på rules->by set->regler i set a.b.., mulighet for å få info mer komprimert, mindre bokser
- Under update->manual update er det en knapp med norsk tekst (Velg fil)