NTNU
Norwegian University of
Science and Technology

Processing of Notifications Produced by Intrusion Detection Systems in CERN's Security Operations Centre

Author
Amund Faller Råheim

Bachelor of Science in Engineering - Computer Science
20 ECTS
Department of Computer Science and Media Technology
Norwegian University of Science and Technology,

22.05.2017

Supervisors
Basel Katt at NTNU
Liviu Valsan at CERN

Sammendrag av Bacheloroppgaven

Tittel:	**Berikelse av Varslinger Produsert ved Oppdagelse av Potensielle Sikkerhetstrusler i CERNs Nettverksinfrastruktur**
Dato:	22.05.2017
Deltaker:	Amund Faller Råheim
Veiledere:	Basel Katt at NTNU Liviu Valsan at CERN
Oppdragsgiver:	CERN - European Organization for Nuclear Research
Kontaktperson:	Liviu Valsan, liviu.valsan@cern.ch
Nøkkelord:	sikkerhet, databerikelse, varsling, inntrengningsdetektering
Antall sider:	34
Tilgjengelighet:	Åpen

Sammendrag: Avhandlingen omhandler utvikling av an applikasjon for å behandle varslinger produsert av inntrengningsdetekterende programvare. Varslingene oppstår når kjente indikatorer på inntrengning detekteres i nettverkstrafikken mellom CERNs interne nettverk og internet. Berikelsen er en del av CERNs operasjonssenter for informasjonssikkerhet, og skal gi behandleren av varslingene et bedre, mer informativt og korrekt overblikk av hendelsen. Applikasjonen henter varslinger i en gitt tidsperiode, behandler de ved å inkludere data som enheter involvert og hvilke trusler som er koblet til denne indikatoren. Til slutt blir varslingene gruppert og sendt som email til de rette personene.

Summary of Graduate Project

Title:	**Processing of Notifications Produced by Intrusion Detection Systems in CERN's Security Operations Centre**
Date:	22.05.2017
Authors:	Amund Faller Råheim
Supervisor:	Basel Katt at NTNU Liviu Valsan at CERN
Employer:	CERN - European Organization for Nuclear Research
Contact Person:	Liviu Valsan, liviu.valsan@cern.ch
Keywords:	security, data enrichment, intrusion detection, alerts
Pages:	34
Availability:	Open

Abstract: This thesis focuses on the implementation of an application processing notifications produced by intrusion detection systems. Notifications are produced upon detection of a known indicator of compromise in the network traffic between CERN's internal network and the internet. The enrichment is a part of CERN's security operations centre, and should provide the data analyst a better, more informative and correct overview of the incident in question. The application retrieves notifications from a time window, processes them by inflating fields to contain data about involved devices and information about the threats that are linked to this indicator of compromise. Finally the notifications are aggregated and sent to the relevant persons by email.

Acknowledgements

I would like to thank all the people who have allowed for this thesis to happen. I am grateful to my supervisor at CERN, Liviu Valsan, for his great expertise in getting this project functioning and deployed in a shorter than expected time.

Furthermore, I would like to thank my NTNU supervisor, Basel Katt, for his assistance with clarifying the expectations of a thesis, and giving me valuable feedback and advice for my thesis.

Thank you to Erik Hjelmås for believing in me, and presenting me with the opportunity to travel to Switzerland for my thesis, and to Stefan Lüders for taking me in to the warm and safe environment of CERN's Computer Security Team.

Lastly I would like to thank all the colleagues and friends I've met at CERN, and in the Computer Security Team in particular. It's been a great pleasure working with you and sharing the struggles. Your support and combined knowledge has helped me learn and grow throughout the past semester.

Contents

1 Introduction

CERN is a highly exposed international research organisation, hosting thousands of researchers and tens of thousands of visitors every year, and thus acts as an attractive target for computer security threats. From active cyber-attacks and continuous vulnerability scanning, to phishing and advanced persistent threats, the threats to CERN's networks and information systems are many.

As part of the work to decrease the number and severity of compromises and lessen their impact on the organisation and their computing infrastructure, the Computer Security Team at CERN has deployed a comprehensive infrastructure to inspect network traffic. It's job is to detect indicators of some of these attacks in close to real time, in order for the team to respond quickly when an incident occurs.

1.1 The CERN Compromise Detection Landscape

The Security Operations Centre (SOC) uses many sources of information. A diagram of it's architecture can be seen in Figure 1. It analyses all network traffic at the outside network perimeter, meaning traffic going between CERN and the internet, such as HTTP transmissions, file transfers and SSH connections. This data is stored, and accessible to the Computer Security Team through various interfaces.

The data is compared to a set of known Indicators of Compromise (IOC) stored in a collective database known as the Malware Information Sharing Platform (MISP) [1]. Various Computer Security Response Teams (CERT) may contribute to this database with indicators such as malicious IP addresses, domains or URLs, and file hashes. CERN controls which CERTs' they want to receive information from in their MISP, and who they share their IOCs with.

Correlation between the data traffic and the IOCs is done by tools such as the Bro Intrusion Detection System (IDS). It produces a notification every time an IOC is detected in the fields of an inspected data packet.

These notifications are of value to the Computer Security Team and serve as alerts when an intrusion has happened, as well as forensic evidence allowing for gathering more intelligence around the event that produced the notification. Their contents are however of low quality to a human analyst, and require a lot of generic look-ups in order to benefit the security analyst.

There is room for improvement at this stage in the infrastructure, and the Computer Security Team wants the notifications to be processed post detection to see correlations and enrich certain data fields in the notifications. This is also seen in the light of a plan to upgrade the Security Information and Event Management (SIEM) system, the rightmost box shown in Figure 1, which will feature a view for IDS notifications.

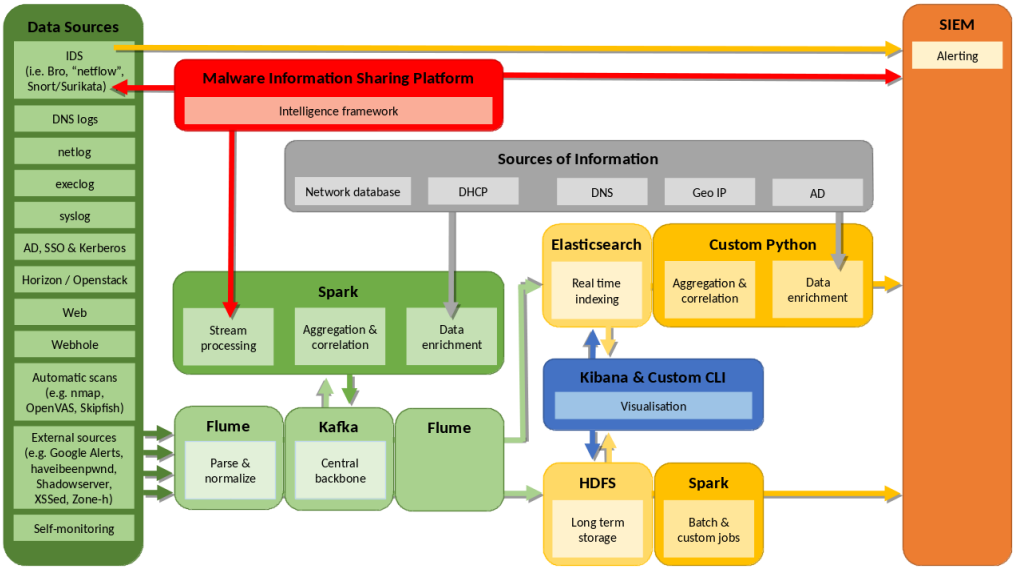

Figure 1: The architecture of the Security Operations Centre, showing data sources, backend software, and frontend interfaces.

1.2 Project Description

The outcome of the project should be a Python application processing notifications produced by intrusion detection systems in CERN's security operation centre infrastructure. It will primarily run at regular intervals on a system maintained by the Computer Security Team as a batch job.

The application should get all notifications in a given time period from an Elasticsearch [2] cluster. The notifications contain such fields as source IP address and port of the connection, the destination IP address and port, the time the notification was created, ID of the connection, and the IOC that triggered the notification. An example of a notification as returned from Elasticsearch can be seen in Appendix A

CERN IP addresses involved as source or destination in a notification should be enriched to contain the device name and hardware address that used the IP address at the time the incident happened. The device name should further point to information about the device available in CERN's network device database. Furthermore, the IOC should be inflated with available data from MISP, and all MISP events containing the IOC in question.

Notifications should be grouped based on certain related attributes. These groups of attributes should show context of how related devices behaved around the time of the incident. Logs for this data is available in various Elasticsearch indices [3], and can be queried using data available in the notifications.

The groups of processed notifications with available data from the enrichment should be emailed to CERN's Computer Security Team to be handled further. For future use cases of these notifications, a data structure with enriched notifications having the same primary keys as before processing should be sent to a dedicated Elasticsearch index. The structure of grouped notifications should also be saved to Elasticsearch in a separate

index.

With the application running as a batch job on a remote environment, the application should be highly configurable for the administrator, without having to change the code. In addition to this the application should work as a command line tool, allowing the user to provide arguments used in the following run.

1.2.1 Objectives

The following objectives are derived for the project. The impact and performance objectives are related to the resulting program, whereas the learning objectives relate to the developer.

Impact Objectives

The result of the notification enrichment and aggregation should be of benefit to the Security Escalation Coordinators (SEC) in the Computer Security Team. They should be able to get a standardised presentation of the notifications, featuring enough data to decide a course of action for the incident. This should result in less time spent on generic incident investigation, and may also give insight into attack patterns for malware and ongoing attacks.

The project should also result in several Python modules for interfacing to data sources used for enrichment. These modules should be generic, in order to be useful for the team in future projects.

Performance Objectives

- Unexpected input should result in a controlled exit and provide feedback on what went wrong;
- Timeouts from services should not result in a crash;
- Output of queries to data sources performed by the application should not result in crashes;
- A failure to perform some parts of the enrichment should have minimal impact on the rest of the enrichment;
- The program should be highly configurable to allow for changes in e.g. API keys and other verification input, extraction window, input indices, format of notifications, and foreign keys for contextualisation.

Learning Objectives

Based on the learning goals in the course description of IMT3912 - Bachelor's Thesis [4], the following learning goals have been proposed:

- Developing secure and reliable Python applications;
- Developing a highly configurable application;
- Interfacing with APIs such as elasticsearch-py, and various SOAP and REST APIs;
- Working in a security motivated environment.

1.2.2 Limitations

The program should:

- Integrate seamlessly into the existing SOC by running at certain time intervals on a CentOS 7 system;

- Be written in Python as encouraged by the Computer Security Team;
- Interface with necessary services through provided frameworks with the minimum privileges necessary to perform the tasks;
- Only depend on Python libraries available from the default OS repositories at CERN, and feature other necessary code.

1.2.3 Scope

The following list describes the scope of the project:

- The application should be tested and operational by the end of May 2017;
- The application should be modular and highly configurable;
- Existing databases and services should be used;
- Read-only accounts will be used for the services to the furthest extent possible.

1.3 Supervision

The project is being conducted by a single developer under the supervision of Liviu Valsan at CERN. Liviu is responsible for the development and maintenance of the SOC, and is on rota as a SEC, placing him in the user group of the application and the resulting data.

Furthermore, the developer is supervised at his university, NTNU, by Basel Katt. Basel will supervise in the writing of this thesis which will be handed in to NTNU at the end of the project.

2 Requirements

2.1 Use Case Diagram

Figure 2 shows the involved actors and general use cases of the program as they stand out in the initial requirements. Initialisation of the program is the only interaction from the user, and given a correct set of preconditions (see Section 2.2) each use case should pass with no further user interaction necessary.

The diagram highlights the pipelined nature of the program, where the user or the scheduler starts the process, and the service actors are called in turn. Note that the use cases "Send email" and "Add to index" are use cases for a scheduled run.

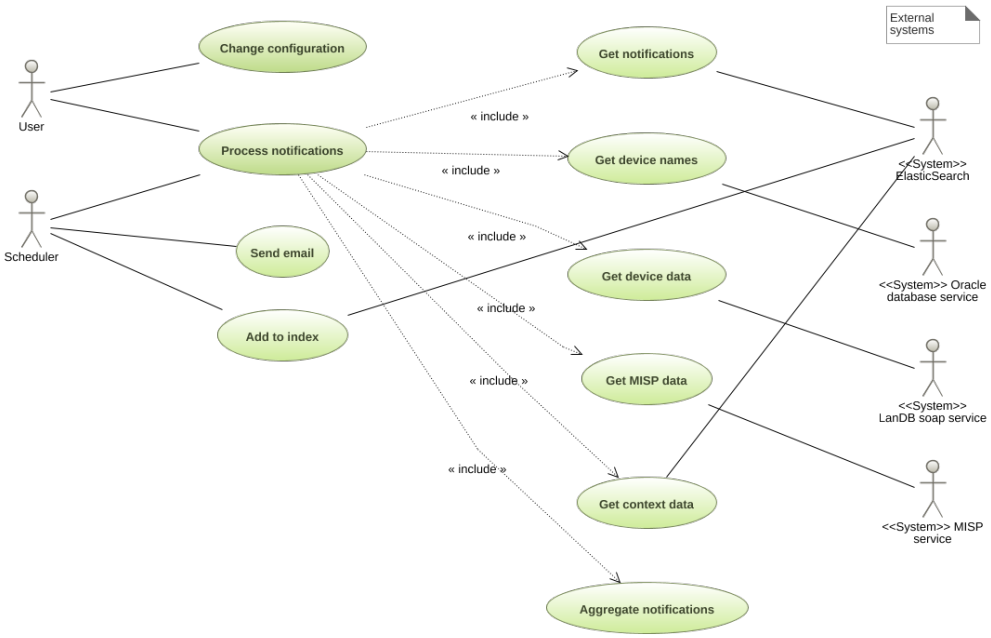

Figure 2: Use case diagram showing the actors and use cases that the system is expected to perform. Made using: GenMyModel

2.2 Use Case Descriptions

Following is an elaboration of each of the use cases present in Figure 2.

2.2.1 Change Configuration

The user wants to change the configuration of the program to tune the parameters for different results, or account for changes in the environment of the application.

Normal flow:

The user does one of the following:

1. Edits an existing configuration file;
2. Overrides one or multiple options in the configuration file through available parameters in the command line interface.

Alternative flows:

If the user provides a wrong option in the command line interface, the program does not run. If one of the options in the configuration file is wrongly configured or not provided, the program might not be able to complete the enrichment. Information on dependencies is provided in preconditions in the use case descriptions.

2.2.2 Process Notifications

The program is configured, and the user wants to see enriched notifications from a given time window. Through the command line interface, the user might enter alternate settings to the configuration file, which will be used in place of the preconfiguration.

Preconditions:

- A configuration file;
- Access to the necessary systems (i.e. by being on the CERN network).

Normal flow:

As seen in the use case diagram in Figure 2 this use case is build up by (*includes*) several other use cases. See the referenced descriptions for more information about these use cases.

1. Configuration is read from a file, and any provided parameters overwrites the values from the file;
2. Notifications are fetched from Elasticsearch indices specified in the configuration file (see Section 2.2.3);
3. Device names for the local IP addresses are fetched from a database with network logs (see Section 2.2.4);
4. Data such as location and person responsible for a device is fetched from CERN's network database (see Section 2.2.5);
5. Data about the MISP attribute (IOC) that triggered the notification as well as the MISP events containing that attribute are fetched from a MISP database (see Section 2.2.6);
6. Context about the connection, any related network traffic as well as the destination and source IP addresses are searched for a given time window (see Section 2.2.7).

Alternative flow:

If the configuration file is not found, enrichment fails.

2.2.3 Get Notifications

The user needs to get the input data from Elasticsearch as a basis for the enrichment.

Preconditions:

The following must be provided in the configuration, or though arguments to the application for successful extraction:

- A URL and a port to the Elasticsearch host;

- Username and password with read access to the input indices;
- One or more Elasticsearch indices for input, with optional search field and value to filter the documents in the respective index;
- A time window for the document extraction;
- The Elasticsearch service must be operational.

Normal flow:

1. Get the authentication data for Elasticsearch, as well as the URL the server web interface;
2. Establish and check connection to Elasticsearch;
3. Get a list of input indices, including any filters that should be applied to each respective index;
4. Get the maximum number of notifications to extract, or extract all if no maximum is provided;
5. Get a time window for the extraction;
6. Get the timeout for the search in number of seconds;
7. Search for documents matching the provided parameters on the given Elasticsearch client.

Alternative flows:

- One or more of the mandatory values are not provided in the configuration or through arguments: The search will fail, and enrichment can not proceed, and logged;
- The server does not respond, or it responds negatively, due to e.g. errors in search filters or input indices: The error is logged and/or printed, and enrichment can not proceed;
- No new notifications: Process exits.

2.2.4 Get Device Names

The notifications contain source and destination IP addresses, at least one of which is an IP address on one of CERN's subnetworks. The user wants to see the device names and hardware addresses of the local IP address extracted from CERN's DHCP and ARP logs.

Preconditions:

- Notifications with local IP addresses (all notifications should have this);
- A list of all local subnetworks, to find which IP addresses are local to CERN;
- The database service must be available.
- Parameters necessary for connecting to the database containing the logs, including:
    - Service name;
    - Username and password.

Normal flow:

1. Get the necessary data to connect to the database;
2. Connect to the database;
3. Find all local IP addresses;
4. Construct a query to find all rows in DHCP and ARP records where IP address is the

local IP address and notification timestamp is in range of start and end time. From the rows, we want start and end time for the record, the IP address, the hardware address and the correct client name;
5. Get all the relevant records from the database;
6. Pick the best result and aggregate with the search parameters based on timestamp and IP address.

Alternative flows:

- One or more mandatory parameters are not provided: device info will not be fetched;
- The database does not respond, or authentication fails: device info is not fetched;
- An input IP address does not have any matching rows: This IP address will not be enriched with data from the database.

2.2.5 Get Device Data

The user wants to see more information about the CERN device(s) involved in a notification, such as device owner, responsible person, location at CERN, and operating system. This information is available in CERN's network database, which has info about all the devices registered on the internal network.

Preconditions:

- For connection to the network database's SOAP [5] service, one of the following is mandatory:
  - path to a file with a valid authentication token;
  - a set of username, password and user type (e.g. "CERN" or "NICE");
- A URL to the SOAP service is needed for both alternatives;
- For the search, one of the following:
  - a device name from the network logs;
  - a static IP address (e.g. a network interface for a server) that has been correctly configured in the network database;

  Search on a static IP address is the least favourable, but can be used if no device name is found;
- The network database's SOAP service is available.

Normal flow:

1. Authentication will either happen with a valid token or with valid credentials. If a token is available, this method is preferred;
2. Get all device names from previous enrichment;
3. Search using device names in batches of up to 25;
4. Aggregate search results with notifications.

Alternative flows:

- The required fields for token validation is not found: Jump straight to credentials authentication;
- The required credentials for authentication is not found: this part of the enrichment

is cancelled;
- Authentication does not succeed or no response from service: Cancel this enrichment;
- No device names from previous enrichment or from the network database when searching with IP address: Nothing more to do, cancelling this enrichment;
- A search on multiple devices fails: Switch to searching on individual devices.

2.2.6 Get MISP Data

All notifications contain an indicator of compromise from one or more MISP events. The user would like to see more information on the IOC and the MISP events where they occur.

Preconditions:

- The notifications need to contain an IOC in a given attribute;
- A regular expression to filter out the IOC, if necessary;
- URL to the MISP service and an API key with read access;
- The MISP service must be available.

Normal flow:

1. Collect the search parameters (the IOCs) and from which notification they came;
2. Get the necessary data for connection;
3. Connect to the given URL with the given API key;
4. Search for all events with the given IOCs;
5. Aggregate the events with notifications based on same IOCs.

Alternative flows:

- No IOCs were found: No MISP enrichment can be performed;
- The parameters for connecting to MISP were not provided, or did not allow for a successful connection: The enrichment is cancelled.

2.2.7 Get Context Data

Bro IDS dumps a lot of network logs into Elasticsearch. Some of these logs may contain information about some of the fields present in a notification. The user wants to decide which notification fields will be used to search and to which indices in Elasticsearch. This should give them context data about where the given search keys were seen in the moments before and after the notification triggered.

Preconditions:

- The configuration file contains a map of keys in the notification to keys in the rest of Elasticsearch, and to which indices those keys should be used;
- The configured key exists in the notification;
- The Elasticsearch connection is still active.

Normal flow:

1. Get the configuration for how searches should be performed;
2. Add the keys that are not in the given notification to a buffer to be searched later;
3. Iterate the notifications and extract the search variables;

4. Search for the context data in Elasticsearch;
5. Add the context data to the notification under a structure where it's clear which search key was used;
6. Search the newly found data for the keys that were buffered, and perform a second search using these new values.

Alternative flows:

- The configuration for a notification could not be found or parsed: Context search for this notification will not proceed;
- No context data found, or search failed: Empty list is added as result.

2.2.8 Aggregate Notifications

The user wants to see which notifications are related based on given attributes. These should be aggregated to form groups of notifications.

Preconditions:

Attributes that will be used for aggregation must be available.

Normal flow:

1. A table is created with groups of all notifications which have a certain attribute in common;
2. Groups which share one or more notifications are merged to larger groups.

Alternative flows:

If the fields for aggregation is not available for a given notification, it is placed in a group of it's own.

2.2.9 Add To Index

The enriched notifications and aggregated groups might be used for various purposes in the future, and on a scheduled run we want to save important data structures in Elasticsearch indices. This should have a standard output format.

Preconditions:

- The Elasticsearch connection is open;
- Output indices are specified;
- Types [3] for these data structures are specified in Elasticsearch;
- The application has write-access to given indices.

Normal flow:

1. The notifications are parsed to fit the Elasticsearch types and converted to a relevant data structure;
2. The data structure is sent to a dedicated Elasticsearch index.

Alternative flows:

No connection to Elasticsearch or no write access to the index: The data structure is dumped to a file.

2.2.10 Send Emails

The user wants an email with data about notifications that have been grouped based on certain attributes. The email should be neatly formatted, with time lines for the contextualised data and highlighting of the data that has been inflated. The structure of the email is subject to change.

Preconditions:

- A receiving email address;
- A sending email address.

Normal flow:

1. Format the data necessary to form emails;
2. Format an HTML body for the emails with all relevant data for each group;
3. Send emails.

Alternative flows:

The program does not check if the emails are successfully sent.

2.3 Quality of Service Requirements

2.3.1 Configurability

Since the program will run multiple times per day, and be deployed on a remote computer, the program should be as agile to changes as possible. Taking input in the form of e.g. credentials, URLs and index names, a configuration file is necessary.

Some of what should be kept configurable requires more complex data structures than a simple field with text. To account for multiple types of notifications with different formats and different parameters for context search, the configuration file will require the user of the program to fill in some configuration fields for each type of notification in input.

2.3.2 Reliability

All exceptions thrown during runtime should be caught, and even when certain parts of the enrichment fails there should be a minimal impact on the total processing. This is a prerequisite to ensure completability.

2.3.3 Completability

The application should process all notifications coming into the input indices. This is necessary if the contents of the output index is to be trusted as a primary source of notifications so this can be used to present the data at a later point. With the applications role as a notifier of the security escalation coordinator through email, it is also quite apparent that it should process all notifications.

2.3.4 Integrity

It is important, but not critical, that the data retrieved by the application is correct. The person responding to the incidents should know the shortcoming of the program, and if the assessment can be done specific cases of uncertainty should be reported to the user.

2.3.5 Performance

The program should finish processing a batch of notifications before the next job is started on scheduled runs.

With a constant influx of notifications to the indices, it should process notifications with real time speed. This is however not expected to be the case, and the notifications can come in large numbers in short time periods. It is hard to assess the performance requirements of the program (e.g. with a notification-to-time ratio) because of the unreliable number of notifications being produced.

2.3.6 Logging

Running as a batch job, the program will not have supervision during a run. Any abnormalities should be logged, so the system administrator can troubleshoot what went wrong. The log should state what the program was trying to do when it failed, and preferably which state it was in when it failed. This mostly includes caught exceptions.

2.3.7 Documentation

Code should be documented with inline comments in the code. Meaning every major ("global") function should be documented with e.g. non-trivial parameters and return values. It is important for the quality of the modules that there is consistency within the modules. This includes commenting as well as code conventions.

Documentation for known weaknesses, bugs or quick-fixes made during development should be handed over to the CERN supervisor at the end of the project. This may be in the form of inline comments.

3 Development Process

3.1 Work Environment

The development process has spanned from the 2nd of February to the 26th of May 2017. It is worth noting that this continues a few days after this document is finalised.

During this time period, the developer has been working in an office with one or two colleagues. These have assisted in problem solving as sparring partners and code reviewers.

The workdays have been at least eight working hours, five days per week. This includes some unrelated activities like seminars, lectures, and weekly meetings for the team.

One of the two weekly meetings include a "round table" discussion where all the present team members are expected to give updates on what they are working on. The developer has received some useful input during these meetings by the full group of future users. Especially the contents of the alert emails, which is the most prominent result of the project, have been discussed.

There have been informal weekly meetings between the developer and the CERN supervisor Liviu Valsan, in which problems that have occurred during development have been discussed and the backlog has been filled and sorted by priority. This has been of great importance for clarifications and to help the developer in formulating his ideas.

Meetings over Skype have been held between the developer and the NTNU supervisor Basel Katt. The main topic of discussion has been the outline of the thesis (this document), and the larger problems and solutions that have been implemented. This has also been useful to unravel the larger problems underlying the application, and what to focus on during the development process.

3.1.1 Routines

The developer has worked with a task board containing the worklog, the items in production, items to be tested and reviewed, and the finished items. This backlog has featured different kind of issues, like requirements and features to be implemented, as well as bugs and "quick-fixes" that needed better solutions in the future.

A whiteboard and a writing pad has been used as supplementation for scribbling down ideas. The contents of these have been summed up as issues in the backlog, or as documentation in this document. They are not considered a part of the final documentation, and will not be handed over unless they cover parts of the application that have not been covered by the official documentation.

Since the developer has evolved and learned a lot in the duration of the project, some of the code might be inconsistent. This is a subject for the code review being performed with the developer and the CERN supervisor.

3.2 Coding Environment

3.2.1 Integrated Development Environment

PyDev for Eclipse [6] was used as integrated development environment (IDE) during the development. The developer had experience with the Eclipse environment, although not with PyDev in particular. After asking colleagues in the Computer Security Team, PyDev was confirmed to be a good choice.

Eclipse is a free and open source IDE with a market for third party plugins, and the PyDev plugin allows for auto completion and some basic refactoring functionality for Python development.

The refactoring functionality for renaming and extracting methods and variables proved useful for the development approach used during the project. The result of the refactoring usually required spending time testing and reviewing the changes. This meant that the time saved by using the refactoring functionality was reduced.

3.2.2 Version Control

CERN's IT department provides an internal GitLab [7] repository hosting platform that is recommended for all internal projects. This recommendation is followed, and Git and GitLab is used for version control.

There have been few commits in certain periods, partially because the code would not be released to production before a large part of functionality was in place, and in parts because committing was forgotten by the developer at certain working iterations of the program.

3.2.3 Issue Tracking and Task Board

Jira [8] is used to issue tracking by the team, and contained an epic for post processing of notifications. The epic contained issues for the use cases described in Section 2.1.

Trello [9] was used to keep track of the backlog and subtasks, refactoring ideas and bugs. Trello was chosen over Jira for this becuase the developer wrongly assumed tracking issues in Jira would result in spam mails for the team. The Trello board quickly grew into a place to document ideas that could not be handled immediately, and thus the board contained dozens of tasks at the end of developing the first version of the program. These issues were mostly fixed in the code review and pair programming session following the first version.

3.2.4 Documentation

Major decisions were documented as inline code and/or on a document on the LaTeX writing platform ShareLaTeX [10]. This document, including relevant figures, application design, and important decisions, will be handed over as part of the final documentation.

4 Technical Design

4.1 Language

Python 2.7 [11] was used as development language for the entire application. The program started with a code base handed down from the project manager that was written in Python 2.7. It was also recommended from the Computer Security Team to use this version of Python. Python version is further discussed in Chapter 7.

4.2 Modules

In order to make the application modular and provide libraries that may be useful for future use, the applications main external interfaces have been separated in four distinct components. Lastly the configuration is handled by a fifth module. The components are tied to the use cases discussed in Section 2.2, and can be seen in Figure 3. The modules have been named after what they interface to.

4.2.1 es.py

This module contains the class `Elasticsearch`. It provides an interface to Elasticsearch, allowing for authentication, simple searches, checking for the existence of a given document, and writing a data structure to an index.

The initialiser for the class takes the arguments necessary for connection to Elasticsearch. In order to establish a successful connection, a host URL and a valid username and password pair is needed. These are saved as class attributes, and the authentication process starts.

Authentication creates a new Elasticsearch client which is saved as a member variable. Authentication happens using HTTP over SSL with SSL certificate check. The client is then pinged to check that the connection is working as expected. A Boolean value is saved as a member variable, indicating if a successful connection was established.

The search function checks that it has received the mandatory arguments and that a connection to the Elasticsearch instance is still open. It constructs a minimally viable search query, of which an example can be found in Appendix B.

The arguments needed for a search are a list of indices in which to search, terms to be searched for as a list of Python dictionary with keys and values or just a single dictionary, and a start and end time for the search. Additionally, there are the following default arguments:

- `size` which defaults to 10. The maximum number of results to get. Value 0 implies "get all" in the given time period;
- `request_timeout` defaults to 30. The number of seconds before the Elasticsearch query times out. If query asks for all items, this is also the timeout of each individual batch queried;
- `occur` defaults to 'should'. Legal values are 'should' and 'must'. If multiple search terms are provided, 'should' will require only one of the terms to match in a docu-

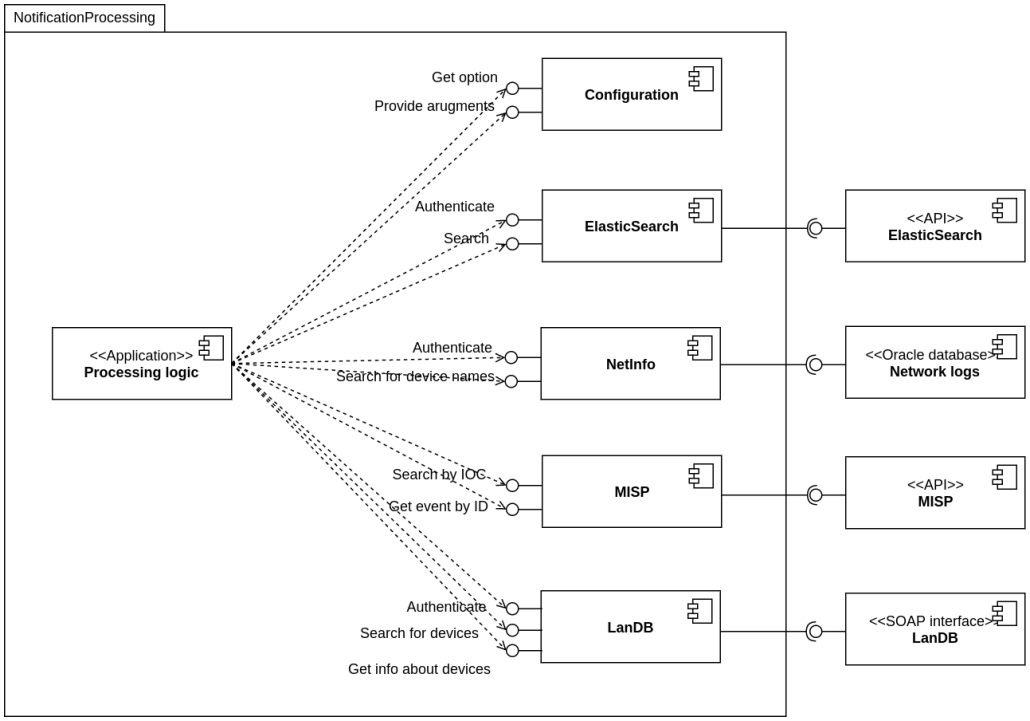

Figure 3: Component diagram. Shows that the notification processing application uses all the interfaces of the modules, and data sources the module relies on. Made using Draw.io.

ment, whereas 'must' will require all the terms to match.

The Elasticsearch indices that this module is meant to interface are partitioned with dates of the residing documents. In order to minimalise the strain on the Elasticsearch service, all the indices that are queried get appended dates based on the start and end time of the search filter.

The module is dependant on the python module Elasticsearch-py [12], a low-level client for Elasticsearch. The versions correspond to versions of Elasticsearch. CERN is currently running Elasticsearch version 5.2.1. Because of the limitations outlined in Section 1.2.2, version 1.9.0 provided by CERN's RPM providers on CERN CentOS 7 [13] is used since it provides the necessary functionality.

4.2.2 landb.py

This module contains the class `LanDB` and the exception class `AuthenticationError`.

The `LanDB` class is initialised with the necessary parameters to authenticate and connect to the network database *LanDB*. Mandatory parameters are URL to the network database's SOAP service, username and password. Furthermore it is possible to provide a location to cache the authentication token, change the "SOAP user type" (defaults to "NICE"), and provide a proxy for the connection. Password authentication is called from the initaliser, but the class also has a function for authentication with a cached token.

The class provides search functions to:

- get list of device names matching provided criteria;

- get info for a number of devices based on a list of device names.

The function to get device info searches in batches of 25 (the maximum batch size). If a search fails, it will switch to searching device info for individual devices.

In order to communicate with the SOAP service, suds, a SOAP interface for Python 2.x, is used. It was support for WSDL [14] which is used by the network database's SOAP service. An encoder [15] is used to fix broken WSDL schemas.

4.2.3 netinfo.py

Contains the class `NetInfo`, which is an interface to an Oracle database service with information about devices. The class is initialised with an Oracle service name, a username and a password, all necessary for connecting to the database.

The module offers a function to search for devicenames based on a list of sets of related IP addresses and timestamps. These values are then parsed to a search query filter and a dictionary with all the prepared values going into the prepared query. The database query used to search the database can be seen in Appendix C. This is further discussed in Section 5.4.2. Because the DHCP and ARP logs may return different results, it is crucial that the correct record is picked. If DHCP and ARP return two different device names, if would mean that one of the two records has an incorrect end time, which encompasses the timestamp used for search. By assuming that the start times of the records are correct, the record with the latest start time is the correct database record.

The module is dependant on cx_Oracle [16], a Python interface module to Oracle databases. Version 5.2.1 is currently provided by CERN's RPM providers, and has been used for development and testing. A read-only account for the necessary tables has been provided for the application and was used for testing.

4.2.4 misp.py

The `misp` module contains the class `MISP`. It is a wrapper over the PyMISP library used to access MISP and query for data. The class is initialised with a URL to a MISP instance and a API key to authenticate.

It provides the following search functions:

- get an event by its event ID.
- search based on provided values in either attributes or events. What to search is provided as a parameter, where attributes is the default.

The module is dependant on PyMISP. This is a library to access MISP using Python. It is NOT provided by the CERN RPMs, and needs to be included in the project from source.

4.2.5 Configuration Handling

Configuration is handled using a class hierarchy. This is is line with both the modularity of the program, as well as the future vision of a scalable SOC. The class hierarchy can be seen in Figure 4. Each class represents a section in the configuration file.

The top level class is `SocLibConfig` (from "SOC library configuration"). All objects of this type share a common state (the "Borg" design pattern [17]). That means, once an object of this type is initiated, new objects of the same type should return the same configuration data unambiguously.

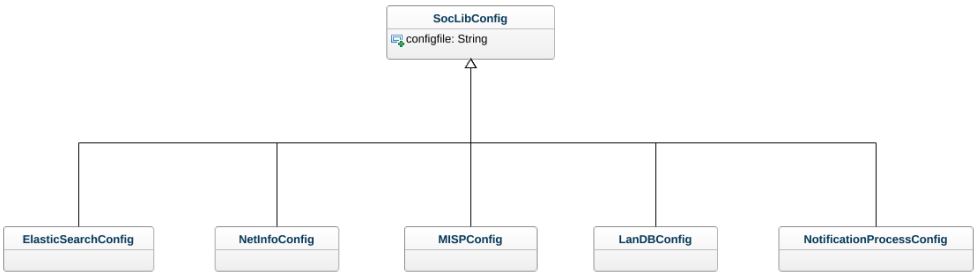

Figure 4: The class hierarchy of the configuration handler, showing the superclass, SocLibConfig and the subclasses belonging to each section of the configuration file.

`SocLibConfig` features common functions to all configuration objects, such as reading a configuration file and overwriting with provided command line arguments. The `__getattr__` function is overridden to get the configuration parameters as member variables. It first checks the command line arguments for the parameter, otherwise tries to retrieve the value from the configuration file. If an object of this type is used, the "general" section of the configuration file is called.

The subclasses represent a section of the configuration file, excluding "general" which is covered by the super class. When getting an attribute from an object of these types the corresponding section will be used. These objects add their related arguments to the command line interface.

If command line arguments for start and/or end times are provided, a function to validate time and date input is called. It tries to parse the dates and/or times provided in the command line to one of accepted formats. If it succeeds, it returns a Python datetime [18] object, unaware of timezone.

5 Implementation

This chapter documents some of the decisions made in the implementation of the program. This is not a complete documentation, but shows some important aspects and challenges of the program.

5.1 Enrichment and Aggregation

This section describes the main logic for extracting search keys, enriching them, and aggregating the notifications used in the application. The functionality described here was important to realise the use cases related to processing the notifications and adding additional information to the notifications.

5.1.1 Inflating the Data

Input indices are queried one at a time from Elasticsearch to account for different search filters. Search parameters for `NetInfo` and `MISP` are prepared as dictionaries with keys produced from the notifications for easy access to the result of the queries. `NetInfo` will only be searched for IP addresses belonging to CERN's subnetworks that are specified in the configuration file.

Fields containing IOCs are parsed using regular expressions if this is provided in the configuration file. For example, the field may contain "Indicator = 123.123.123.123", which would return the IOC "123.123.123.123" when matched with the regular expression `^(?:Indicator = )?(.*)`.

After getting the MISP attributes for all IOCs from MISP, the event IDs are extracted from the attributes and MISP is searched again for data on all the found events.

5.1.2 Aggregating the Notifications

Grouping notifications is an important enrichment use case in order to lowers the number of emails being sent. This should be done in an optimal way, so aggregating with nested iteration loops of the notifications is not a good alternative. Rather, a hash table (using a Python dictionary) is used to group the notifications based on various values.

Fields used for placing notifications in the hash map are IOC, MISP event ID, device name, hardware address and IP addresses. The quality of aggregations of fields relating to an IP address is possible to rank. Firstly, if a device name is available for an IP address, this is a better option than a hardware address or an IP address. Secondly, if a hardware address is available, this also unambiguously shows which device is behind an IP address. The weakness is that a device using multiple network interface cards could operate using multiple IP addresses and multiple MAC addresses, whereas a device name would encompass this. Lastly, an IP address would give a less reliable link. If the IP address is a static one for, e.g. a web server, this could be as reliable as an aggregation on device name. If it is a dynamic IP address, there would be no way to know if the IP address has been leased to different devices in the time passed between the notifications.

Notifications should contain two IP addresses for source and destination of network

traffic. Consequently the notifications are placed in two groups based on source and destination host (device name, hardware address or IP address), at least one based on IOC and at least one based on MISP event. These groups need to be merged.

All the groups are operated as sets, meaning they have no duplicates. With a motivation for further aggregation, the sets are checked for intersects. Aggregating the groups by iterating them and iterating them again for each group would require numerous iterations, and would require repeated grouping untill no intersects are found between any groups.

Instead the groups are merged by placing the results in a new list of groups. By iterating all initial groups (the "merging group") and checking them only against the new list, the result is an algorithm which takes available shortcuts, and becomes faster with more grouping. It would only have to iterate the initial set of groups one time. All groups in the new list are merged that have an intersect with the merging group.

Following is pseudo code for such an aggregation function, where intersect and union are the mathematical functions $\cap$ and $\cup$.

```
for group in groups:
    for aggregated_group in aggregated_groups:
        if (group intersect aggregated_group) is not empty:
            if group is not previously aggregated:
                aggregated_group = aggregated_group union group
            else:
                previously_aggregated_group = previously_aggregated_group
                                              union aggregated_group
                                              union group
                aggregated_group = empty set
remove empty sets from aggregated_groups
return aggregated_groups
```

Note that the latest iteration of the program only groups notifications based on MISP events. This is to avoid certain groups being very comprehensive. The planned security information and event management system may feature a view of notifications where grouping on different attributes is interesting. The code for grouping is thus still of future interest.

5.2 The Configuration File

This section highlights some of the choices made when implementing the use case of providing configuration to the program. The program is initiated either through the command line by a user or as a scheduled job. Command line arguments can be provided in both cases, featuring autocompleting functionality for the argument names in a bash shell. The application reads the configuration file and overwrites the option values with the command line arguments.

5.2.1 Special Fields

The `config` module described in Section 4.2.5 has the job of parsing more complex data structures in the configuration file. The motivation for and implementation of these data

structures will be elaborated in this section. Note that these data structures can not be overridden using command line arguments.

Because the program relies on correctly identifying certain fields in the notifications, and because notifications are expected to change, these fields need to be specified in the configuration file. This includes IP addresses that will be enriched, the timestamp of the event and the IOC. An optional regular expression can be added for the IOC, as described in Section 5.1.1.

One solution for supplying field-to-type specifications would be to use long and advanced field names which can be reconstructed from a notification. This would mean a configuration field reflecting both the notification type and the field type. The solution would allow the administrator to inject values using arguments in the command line interface. However, since the program relies on having both input indices, search terms, and field specification related to a notification type this would mean a lot of mandatory arguments which are dependant on each other.

The implemented solution is to use a data structure to contain multiple configuration data under one field. Since the field specification is related to the type of the Elasticsearch document [19], it is handy that an index unambiguously contain only one document type. This means the input indices and their search keys should be stored in the same data structure as the specification of fields.

The final data structure is a dictionary where the keys are index names and search terms and field specification are values. New indices with notifications are not expected to occur often, and it is of little consequence that they can only be added in the configuration file, and not through the command line interface.

All of CERN's subnetworks are listed in the configuration file. This is a list of strings with subnetworks in CIDR notation [20]. These are not expected to change very often, and are not considered part of the use cases requiring testing with the command line interface.

5.3 Timezones and Timestamps

Timestamps occur in three ways in the application: as local times with daylight savings time and time zone specification, as UTC standard time, and as Unix (or "epoch") time.

Timestamps from Elasticsearch are in standard UTC time. However, when searching using `NetInfo`, local time (in CERN's time zone) is needed for the ARP logs and Unix time for DHCP. All output should be in local time as specified in the configuration file.

User input is assumed to be in local time and are converted to localised Python datetime objects, and made time zone and daylight savings time aware. When querying Elasticsearch for notifications they are converted to UTC time. The timestamps of the notifications are then converted to localised datetime objects.

`NetInfo` requires the provided datetime objects to be timezone aware, in order to successfully separate between local time for the ARP logs and a Unix timestamp for the DHCP logs. The start and end times in the result of the search are converted to time zone aware datetime objects before they are returned.

5.4 Searching With NetInfo

`NetInfo` is querying database records from DHCP and ARP logs in order to find which device used a given IP address at a given time. Retrieving the correct data is important for the integrity of the enriched data.

5.4.1 Input and Output Datastructure

`NetInfo`'s search function requires timestamps and IP addresses from the same notification. These are passed as a hash table with reproducible keys derived from the IP address and the timestamp. It ensures that no duplicates occur among the search parameters in the case that an IP address has been involved in two notifications at the same time.

The values of the hash table contain the search keys which work as the basis for aggregation between the input and the search result from the database. The function returns the input dictionary with the additional data added to the correct input dictionary items.

Here is an example of a printed output data structure from a search with `NetInfo`.

```
{'2017-01-01T00:00:00.000000 123.123.123.123': {
    'timestamp' : datetime(2017, 01, 01, 00, 00, 00, 000000,
                           tzinfo = 'Europe/Zurich'),
    'ip': '123.123.123.123',
    'hwaddr': '06-00-00-00-00-00',
    'device_name': 'mydevice',
    'start_time': datetime(2016, 12, 31, 23, 55, 00, 000000,
                           tzinfo = 'Europe/Zurich'),
    'end_time': datetime(2017, 01, 01, 00, 05, 00, 000000,
                         tzinfo = 'Europe/Zurich'),
    'src': 'dhcp'
    }
}
```

The additional data from the enrichment are found under the keys:

- `hwaddr`:the hardware address of the device that was using the IP address;
- `device_name`: device name from the log, as is will be found in the network database;
- `start_time`: for DHCP this is the time when the IP address was assigned to this host, while for ARP the time when a connection using this combination of hardware address and IP address was first seen;
- `end_time`: when the IP address will be invoked by DHCP, or when the hardware address stopped using this IP address;
- `src`: either "dhcp" or "arp".

5.4.2 The Database Query

To achieve higher performance and avoid multiple queries to the database, a single query is constructed for all the necessary information. An example of the query can be seen in Appendix C. For the security of the database, the query is prepared with placeholders for all the parameters and the search values are provided upon execution of the query.

The query is a union of two selects, both querying a start and an end time, the IP

address provided as search parameter, the source of the data (as "dhcp" or "arp"), the device hardware address, and the device name.

The DHCP logs may have two device names for one IP address at a given time, but only one hardware address. One of the device names will be the name provided by the device when it requests an IP address from DHCP. The other name is the name that is registered in CERN's network database as the name of the device. This is the correct name for the device for further use. By joining the hardware address from the network logs with a table containing hardware addresses and device names as seen in the network database, it is ensured that we get a device name that we can use to search the network database later.

5.5 Sending Emails

Each email features one group of notifications and all related MISP events. In order to keep the information structured and comparable, the notifications and events are placed in two respective tables. This way all values from the same data field are in the same column, and descriptive column names can be provided.

HTML is used for formatting of the emails as this allows for links and style encoding in the mails. The Python library Jinja2 [21] was used for the email template. It allows for logic (like loops and conditional statements) and Python function calls inside the template. This way it is easy to create a table of notifications, by simply iterating over the notifications and MISP events respectively and producing columns using a list of keys to the notification or MISP event.

6 Testing and Quality Assurance

6.1 Code Review and Quality Assurance

Code review and testing has been performed involving the CERN supervisor in the scope of the last three weeks of the project. Using a method of code review similar to pair programming, the program was reimplemented based on the proof of concept which had been developed up to this point.

By abandoning the idea of keeping everything enclosed in a single data structure and rather using hash tables, the performance of the program was increased from the proof of concept application. This reduced the number of nested iterations and allowed for optimised lookup with keys that would better handle a large number of notifications.

6.2 Testing

The modules mentioned in Section 4.2 were reviewed and tested first. The tests were done method by method by providing them with input, printing the output and further analysing the output to see that it produces expected results. Deviations from expectations were investigated. For the `NetInfo` module, the query was cross-checked and tested in Oracle SQL Developer [22]. After testing the modules, the logic for processing notifications was implemented. This included adding a command line interface to inject commands as arguments to the program.

Configuration and command line arguments were first tested by printing options to confirm that they were correctly read from file and overwritten by command line arguments. Further, each section and option was tested in parallel with the modules that required them.

The interface modules from Section 4.2 were all tested first for connection to their respective interfaces. After that, searching functions were tested. For the `netinfo` module a query was constructed to get the desired data. It was then simplified to be a minimum viable query, and made generic to include passed parameters. In the `landb` module, searching for device info in batches of up to 25 devices was tested, and it was verified that it also worked when a batch failed and individual searches were being used.

While testing the query of the `netinfo` module, it was found that devices configured with static IP addresses, which includes important servers in CERN's data centre, did not show up in the DHCP logs, although they did lease IP addresses. This was due to a decision in the configuration of the databases, and will be outside the scope of the project to solve. It does however introduce a weakness to the program, where some local IP addresses can not be resolved to device names using this module.

During testing it was discovered that the modules did not account for differences in time zones when NetInfo did not return the expected data from ARP, and thus multiple local IP addresses were not enriched. This resulted in the changes elaborated in Section 5.3.

Testing the program on a large number of notifications by using a long time span

showed that grouping based on MISP events would in some cases encompass a very large number of events in the same group. This was because some MISP events featured a very large number of IOCs, whereby they also consumed a lot of the notifications. By introducing a whitelist of IP addresses which were not grouped on (e.g. DNS servers) some of these groups dispersed. A work-around ensured that groups aggregated by MISP events with a very high number of associated IOCs were isolated, and could not be merged with other groups. This solution will be further tested and may change on account of user feedback.

7 Improvements

By the end of the scope of this project, the program still has room for improvement. The identified improvements are further discussed here.

7.1 Functionality

The developed application lacked some features which were present in the proof of concept, such as a search for context in the security logs and information from the network database. Furthermore, multiple improvements were proposed by the user group, after an initial test batch of twelve emails were sent to the relevant users. The application and the resulting emails were also the subject of discussion at one of the weekly meetings of the Computer Security Team.

On the basis of this, the following list of proposed improvements are compiled:

- A local IP address which has not been enriched with a device name could be enriched by searching in the history of the network database using the IP address and the timestamp. This would ensure that the enriched data is correct for the time of the notification, even if the device has changed statically configured IP address after the incident. Additional data about a device should also be searched in the historical data. This would ensure that e.g. the person responsible for a device was indeed so at the time of the notification;
- The subject of the email was undescriptive, containing only the number of notifications and the ID of the group. Ideally it should contain a description of the type of MISP events that the email contains, and thereby be an indicator to the type of attack that has been discovered;
- Additional network traffic for a connection queried from the network logs available in Elasticsearch should be provided as a part of the email, allowing for a quick overview of e.g. if a payload has been delivered as part of the intrusion;
- Notifications which have no related traffic should not be sent by email, as they most likely are not related to an infection of a device. They should still be enriched and saved for future reference;
- A column of the table of notifications in the email could show whether a device involved in a notification has any openings in the firewall;
- A table column in the email showing whether a given IOC (e.g. a domain name) has been blocked from CERNs networks should be added;
- Initially, the notifications should be processed and a batch of emails be sent hourly.

7.2 Tools

During the development phase, a couple of experiences were made with accounts to the tools used for the program.

Python 2.7 was used for the project. This was chosen on the basis of a recommendation from the project supervisor. Later in the project, there turned out to be multiple

reasons for picking Python 3. PyMISP recommends using the newest version of Python upon initialisation of the library. In addition all other packages used for the project are available for Python 3. It is possible that the project will be converted to Python 3 in the future, but that is outside the scope of this project.

Elasticsearch uses a domain specific language (DSL) [23] based on JSON for queries through the API. This is tedious to write and requires a lot of correct syntax, which makes it prone for typing errors. An Elasticsearch DSL library [24] is made for Python, which makes this a lot simpler. This could have been used in stead of hand written queries. The downside of this is introducing a new dependency to the program from a library that is not available in the default package repositories for CERN CentOS 7, which the application will be deployed on.

8 Conclusion

The developed application processes notifications on an hourly basis and produces and sends emails, providing the Computer Security Team with a better view of the incidents behind the notifications. It provides a good basis for configuration of enhancements to fit the working patterns of the Computer Security Team in the future when the program is put in full use.

Most of the requirements in Section 2.3 have been satisfied, like having comprehensive configuration options, enriching IP addresses with device information, enriching IOCs with MISP events and sending emails with the most important data. Furthermore, the project has resulted in the five modules mentioned in Section 4.2, which were put to use by other projects in the team immediately after their release.

The quality of service requirements from Section 2.3 were met to a satisfactory degree. Configurability was reached to a necessary degree, and make room for a wide array of changes without changing the code. Although the reliability of the program has not been tested, the program has run reliably with the varied and comprehensive data that has been used during testing. By having invested a lot of time testing and verifying the results, particularly from the network logs, the integrity of the data is as good as it can be.

Assessing the run time of the program is hard, as it relies on uncontrollable factors such as caching in the databases. Nonetheless the program has empirically showed to be able to process around two hundred notifications in a couple of minutes. This is more than enough for the initial configuration of hourly processing.

With verbose and descriptive error, warning and debug messages, the program produces a log which will come in handy for trouble shooting the program. The only documentation available are the inline comments, and what can be derived from this document. The inline comments are however written in complete sentences, and describe in good detail the different data structures and larger chunks of code.

The two most notable requirements that are not part of the final application are saving the data for future use in an Elasticsearch index, and finding context such as network traffic for a notification. It is expected that the application will be further developed in the future, and will most likely encompass these features. Chapter 7 goes into details of improvements for future versions of the program.

Developing the proof of concept proved to be a challenge, that took longer than expected. More communication and code review with the supervisor early in the project could have been beneficial to steer in the direction of the final application from the beginning. It would have been beneficial to spend more time adjusting the application for user feedback, and this will now be up to the Computer Security Team.

First and foremost I have achieved valuable experience with developing secure Python applications during this project. I have learned to develop for configurability and modularity in my programs, and to use APIs and online interfaces for querying data.

Bibliography

[1] MISP - Malware Information Sharing Platform and Threat Sharing - Open Source TIP. (Visited May 2017). URL: http://www.misp-project.org/.

[2] Elasticsearch, RESTful, Distributed Search & Analytics. (Visited May 2017). URL: https://www.elastic.co/products/elasticsearch.

[3] Index vs. Type. (Visited May 2017). URL: https://www.elastic.co/blog/index-vs-type.

[4] IMT3912 - Bacheloroppgave - IMT. (Visited January 2017). URL: http://www.ntnu.no/studier/emner/IMT3912/2016/1#tab=omEmnet.

[5] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). (Visited May 2017). URL: https://www.w3.org/TR/soap12/.

[6] PyDev. (Visited May 2017). URL: www.pydev.org.

[7] Code, test, and deploy together with Gitlab open source git repo management software. (Visited May 2017). URL: https://about.gitlab.com/.

[8] JIRA Software - Issue & Project Tracking for Software Teams | Atlassian. (Visited May 2017). URL: https://www.atlassian.com/software/jira.

[9] Trello. (Visited May 2017). URL: https://trello.com/.

[10] ShareLaTeX, Online LaTeX Editor. (Visited May 2017). URL: https://www.sharelatex.com.

[11] Python 2.7.0 Release | Python.org. (Visited May 2017). URL: https://www.python.org/download/releases/2.7/.

[12] Python Elasticsearch Client. (Visited May 2017). URL: http://elasticsearch-py.readthedocs.io/en/master/.

[13] CC7: CERN CentOS 7. (Visited May 2017). URL: http://linux.web.cern.ch/linux/centos7/.

[14] Web Services Description Language (WSDL). (Visited May 2017). URL: www.w3.org/TR/wsdl.

[15] Schema for the SOAP/1.1 encoding. (Visited May 2017). URL: http://schemas.xmlsoap.org/soap/encoding.

[16] cx_oracle. (Visited May 2017). URL: oracle.github.io/python-cx_Oracle/.

[17] Singleton? We don't need no stinkin' singleton: the Borg design pattern. Visited May 2017. URL: http://code.activestate.com/recipes/66531-singleton-we-dont-need-no-stinkin-singleton-the-bo/.

[18] 8.1. datetime — Basic date and time types. (Visited May 2017). URL: https://docs.python.org/2/library/datetime.html.

[19] Types and Mappings. (Visited May 2017). URL: https://www.elastic.co/guide/en/elasticsearch/guide/master/mapping.html.

[20] Classless Inter-Domain Routing, CIDR notation. (Visited May 2017). URL: https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing#CIDR_notation.

[21] Welcome to Jinja2. (Visited May 2017). URL: jinja.pocoo.org/docs/2.9/.

[22] Oracle SQL Developer. (Visited May 2017). URL: http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index-097090.html.

[23] Query DSL. (Visited May 2017). URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html.

[24] ElasticSearch DSL. (Visited May 2017). URL: https://elasticsearch-dsl.readthedocs.io/en/latest/.

A Elasticsearch Notification

This is an example of a notification returned from an Elasticsearch index. Some fields are removed, and contents have been changed.


```
{
"_index": "bro_notification-2017.05.17",
"_type": "bro_notification",
"_id": "jcqJAXLav90ywDev",
"_score": null,
"_source": {
    "msg": "Intel hit on 123.123.123.123 at Conn::IN_RESP",
    "note": "Intel::Notice",
    "sub": "Indicator = 123.123.123.123",
    "srcip": "111.111.111.111",
    "cuid": "CDnD0PGKo5ez4vWtXg",
    "dst": "123.123.123.123",
    "src": "111.111.111.111",
    "dstport": 80,
    "dropped": false,
    "type": "bro_notification",
    "p": 80,
    "@timestamp": "2017-05-17T00:00:00.001Z",
    "proto": "tcp",
    "srcport": 60000,
    "dstip": "123.123.123.123",
    "actions": [
        "Notice::ACTION_EMAIL",
        "Notice::ACTION_LOG"
    ]
}
```

B Elasticsearch Query

Example With "occur" = "should"

In this example, the search query is created with the following parameters:

- Search terms = {'key': 'value',
  'key': 'another_value',
  'another_key': 'another_value'}
- Start timestamp = datetime.datetime.now() - datetime.timedelta(hours = 1)
- End timestamp = datetime.datetime.now()
- Search occurrence type = 'should'

With occurrence set to "should" and minimum_should_match set to 1, at least one of the terms must be present in a document.

```
{'query': {
    'bool': {
        'filter': {
            'minimum_should_match': 1,
            'range': {
                '@timestamp': {'gte': '2017-05-09T16:40:41.544673',
                               'lte': '2017-05-09T17:40:41.544698'}
            },
            'should': [{'term': {'key': 'value'}},
                       {'term': {'key': 'another_value'}},
                       {'term': {'another_key': 'another_value'}}
                      ]}}}}
```

Example With "occur" = "must"

In this example, the search query is created with the following parameters:

- Search terms = {'key': 'value',
  'another_key': 'another_value'}
- Start timestamp = datetime.datetime.now() - datetime.timedelta(hours = 1)
- End timestamp = datetime.datetime.now()
- Search occurrence type = 'must'

With occurrence set to "must", all the terms must be present in a document.

```
{'query': {
    'bool': {
        'filter': {
            'range': {
                '@timestamp': {'gte': '2017-05-09T16:40:41.544673',
```

```
                            'lte': '2017-05-09T17:40:41.544698'}
            },
            'must': [{'term': {'key': 'value'}},
                     {'term': {'another_key': 'another_value'}}
                    ]}}}}}
}
```

C Database Query For Device Info From Network Logs

The query for the netlog database when input is:

```
{'2017-01-01T00:00:00.000001 123.123.123.123': {
    'ip': '123.123.123.123',
    'timestamp': datetime.datetime(2017, 1, 1, 0, 0, 0, 1)},
 '2017-02-28T16:20:00.000000 111.111.111.111': {
    'ip': '111.111.111.111',
    'timestamp': datetime.datetime(2017, 2, 28, 16, 20)}}
```

Table and attribute names have been changed, and the selects have been simplified.

```
(SELECT start_time, end_time, ip_address, dhcp_log.hardware_address,
        'dhcp' AS source, network_database.network_device.device_name
FROM dhcp_log
    LEFT OUTER JOIN network_database.hardware_address
    ON network_database.hardware_address
         = dhcp_log.hardware_address
    LEFT OUTER JOIN network_database.network_device
    ON network_database.hardware_address.device_id
         = network_database.network_device.id
WHERE (start_time <= 1483225200
       AND (end_time IS NULL OR end_time >= 1483225200)
       AND ip_address = '123.123.123.123')
       OR (start_time <= 1488295200
       AND (end_time IS NULL OR end_time >= 1488295200)
       AND ip_address = '111.111.111.111'))
UNION
(SELECT valid_from, valid_to, ip_address, hardware_address,
        'arp' AS source, network_database.network_device.device_name
FROM arp_log
    LEFT OUTER JOIN network_database.hardware_address
    ON network_database.hardware_address.hardware_address
         = hardware_address
    LEFT OUTER JOIN network_database.network_device
    ON network_database.hardware_address.device_id
         = network_database.network_device.id
WHERE (valid_from <= '2017-02-28 16:20:00'
       AND (valid_to IS NULL OR valid_to >= '2017-02-28 16:20:00')
       AND ip_address = '123.123.123.123')
       OR (valid_from <= '2017-02-28 16:20:00'
       AND (valid_to IS NULL OR valid_to >= '2017-02-28 16:20:00')
       AND ip_address = '111.111.111.111'))
```