Master's Programme in Computer, Communication and Information Sciences

Scalable Honeypot Monitoring and Analytics

Mariia Kovtun

Aalto University

MASTER'S THESIS

Aalto University
MASTER'S THESIS 2018

Scalable Honeypot Monitoring and Analytics

Mariia Kovtun

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.
Otaniemi, 8 June 2018

Supervisor: professor Tuomas Aura
Advisor: Andrew Paverd, Ph.D.

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Author
Kovtun Mariia

Title
Scalable Honeypot Monitoring and Analytics

School School of Science

Master's programme Computer, Communication and Information Sciences

Major Security and Cloud Computing **Code** SCI3084

Supervisor professor Tuomas Aura

Advisor Andrew Paverd, Ph.D.

Level Master's thesis **Date** 08 June 2018 **Pages** 53 **Language** English

Abstract

Honeypot systems with a large number of instances pose new challenges in terms of monitoring and analytics. They produce a significant amount of data and require the analyst to monitor every new honeypot instance in the system. Specifically, current approaches require each honeypot instance to be monitored and analysed *individually*. Therefore, these cannot scale to support scenarios in which a large number of honeypots are used. Furthermore, amalgamating data from a large number of honeypots presents new opportunities to analyse trends.

This thesis proposes a *scalable* monitoring and analytics system that is designed to address this challenge. It consists of three components: monitoring, analysis and visualisation. The system automatically monitors each new honeypot, reduces the amount of collected data and stores it centrally. All gathered data is analysed in order to identify patterns of attacker behaviour. Visualisation conveniently displays the analysed data to an analyst.

A user study was performed to evaluate the system. It shows that the solution has met the requirements posed to a scalable monitoring and analytics system. In particular, the monitoring and analytics can be implemented using only open-source software and does not noticeably impact the performance of individual honeypots or the scalability of the overall honeypot system. The thesis also discusses several variations and extensions, including detection of new patterns, and the possibility of providing feedback when used in an educational setting, monitoring attacks by information-security students.

Keywords honeypot, monitoring, logging, analytics, clustering, patterns

Contents

1. Introduction

Honeypots collect important security information, which can be used for both research and production purposes, helping to detect unauthorized access and investigate the behaviour of users attacking the honeypot.

Honeypots differ from each other by their software and hardware implementation, emulation of different types of systems and management. However, their main purpose is still being able to collect information about attacks. That makes the collection of logs essential for all types of honeypots.

The further analysis of actions inside honeypots allows security analysts to take decisions about the right defense against the detected attacks and to reveal new types of attacks and vulnerabilities, producing value from the honeypots.

1.1 Problem overview

In a basic honeypot system, data is collected from one or a small amount of honeypots. Scalability has been considered in the literature, however, the focus of such discussion is typically in instantiation and routing, rather than monitoring. Meanwhile, *scalable* honeypots systems, consisting of a large number of honeypots, pose new issues to the monitoring and analytics.

First, every new honeypot instance should be configured to collect and route the log data correctly. Second, running a large number of honeypots would produce a significant amount of logging information, which means more traffic and more storage space required. Furthermore, that would produce more noise with the redundant information for the analysis, and make the process of the analysis complicated and time-consuming for the analyst.

All these issues require a new approach for building the monitoring and analytics system, without reducing the efficiency of using honeypots and their scalability.

1.2 Thesis process

To address the challenge described above, this thesis proposes a new scalable monitoring and analytics system for honeypots. Specifically, it achieves this by:

- Defining a set of requirements for the monitoring and analytics system.
- Proposing the design of the system that meets these requirements.
- Describing the implementation of the system according to the proposed design.
- Evaluating the design and implementation against the defined requirements.
- Discussing the possible applications of the system, and comparing it against related work.

1.2.1 Solution overview

In this thesis, the monitoring and analytics system is described, which has the following features for solving the mentioned above issues.

The process of logging is customised to produce only logs of a certain malicious activity and corresponding errors occurring inside a honeypot. This decreases the amount of produced traffic, whilst retaining enough accurate data for the further analysis and detection of patterns of attackers.

The monitoring system has a centralised gathering of logs, allowing running of new honeypots without requiring to produce any changes in the configuration of the system.

Analysis of accumulated data is automated to support the work of the analyst. A new component, called the *analytics engine*, parses all data, divides it by the type of the attack, adds labels to logs for which patterns are known by the occurring error, and clusters remaining data.

The analysed data is visualised in a dashboard that displays identified patterns of attacker behaviour.

1.3 Research scope and goals

The aim of this project is to investigate how to monitor and analyse data from a large number of honeypots, without reducing the scalability of the system. The

scope of the thesis does not cover the design of a honeypot system.

The research questions of the thesis are the following:

RQ1: What requirements should a scalable honeypot monitoring and analytics system meet, and how should it be evaluated?

RQ2: What type of system architecture should the monitoring system use, and how would this be implemented?

RQ3: How can the analytics engine effectively analyse large amounts of data?

RQ4: What are the possible real-world applications of the monitoring and analytics system?

This thesis uses an experimental methodology to answer the above research questions.

1.4 Structure of the thesis

The remainder of the thesis is structured as follows: Chapter 2 describes the background. Chapter 3 presents the system model and requirements to it. A real-world case study is described in Chapter 4. The system design and implementation are presented in Chapters 5 and 6 and the evaluation of the system in Chapter 7. Chapter 8 discusses the solution and its possible extensions. Chapter 9 presents related work. Chapter 10 contains the conclusion.

2. Background

2.1 Honeypots

The idea of honeypots is not new and was first presented in 1986 by Clifford Stoll [36]. He described a system that allows the intruder to access the resource and records actions of the intruder instead of trying to prevent the access. Since then a long history of research has been produced, providing several definitions of the "honeypot" term. In the definition proposed by Pouget et al. [31] in 2003, "A honeypot consists of an environment where vulnerabilities have been deliberately introduced in order to observe intrusions".

The way in which the honeypot is built and how information gained from it is used define the role of the honeypot in the organisation. In general, honeypot detects hackers and captures their actions. Its advantage is manifested in the ability to record almost only malicious activity. The possibility of false negatives tends to zero, as honeypots are configured such way that only intruders scanning the network would try to access them. Honeypots also solve the issue with false negatives, as they do not require knowledge of attacks signatures, recording all activity of hackers, including new types of attacks. Thus, honeypots produce a relatively small amount of information with high value to cyber security.

At the same time, there are some weaknesses inherent to honeypots. First, honeypots can introduce additional risk to their environment. Once hacked, they can be used to attack other systems. Second, honeypots are able to capture only attacks directed against them. Furthermore, the existence of honeypots can be disclosed by fingerprinting. That is, an attacker may discover that it is a honeypot by observing certain behaviour or characteristic [35]. These features should be taken into consideration when making a decision about deploying a honeypot in an information system.

2.1.1 Classification of honeypots

Honeypots can be classified by the purpose of usage to:

- *research honeypots,* honeypots that aims to explore the behaviour of an attacker and to investigate new tools for exploiting vulnerabilities and new types of attacks;

- *production honeypots,* used directly to improve the defence of an information network in the organisation. They contribute in the detection of attackers and their distraction from real resources [26].

  By the level of interaction, honeypots are divided to:

- *low-interaction honeypots* that emulate the server to be attacked, but have no real operating system to be accessed by the attacker. These honeypots often appear as listeners and aim to detect the sources of unauthorized activity. They are able to capture a narrow type of activity, introducing low level of risk to the network where they operate. An example of a low-interaction honeypot is the Monkey-Spider project, where a low-interaction honeyclient emulates a web-browser to crawl web-sites to explore their threats to clients [17];

- *medium-interaction honeypots* that emulate some services in a more sophisticated way than low-interaction honeypots but still do not provide full interaction with the system. Compared to low-interaction honeypots that implement network protocols, medium-interaction honeypots emulate application responses for incoming requests;

- *high-interaction honeypots* that represent a fully functional systems that can be compromised by an attacker. They aim to capture as much information about the methods of attack as possible, introducing a high level of risk to an organisation that uses them. This is the most sophisticated type of honeypots to implement. For instance, HoneyBow collects autonomous spreading malware using high-interaction honeypots [44].

2.1.2 Docker

Docker is a program that performs virtualisation on a level of an operating system running containers [9]. The scheme is presented in Fig. 2.1.

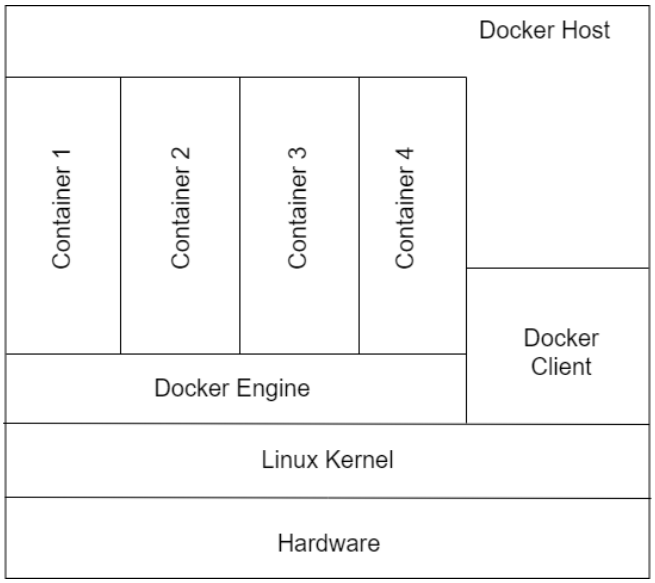

Figure 2.1. Docker on a physical Linux server (adapted from [11]).

Docker runs on most Linux distributions, minimises the usage of CPU and RAM and isolates applications from each other. Docker allows running of independent containers that are more portable and efficient than virtual machines (VMs), avoiding the overhead of maintaining and starting VMs.

Docker Engine is an application with a client-server model, consisting of a three components:

- daemon process (`dockerd`) that represents a server;
- REST API;
- command line interface (CLI) client.

The Docker client communicates with the Docker daemon using a REST API. They can be run on the same server or be connected remotely. The Docker daemon manages Docker containers (performing distributing, running and building) and other Docker objects, including images, volumes and networks.

A Docker image is a template containing instructions for building Docker containers, allowing shipping and storage of applications. Images can be based on other images with additional customisation.

A Docker container is a runnable instance of a Docker image, which can be created from the image and started with configuration options, stopped, and deleted at any time. A Docker image can also be created from the current state of a container.

The ability to run processes in isolation using lightweight containers makes Docker suitable for building honeypot systems. This case is considered in a range of literature [14, 15, 19] and, for instance, is implemented in elastichoney [40] that is intended for catching attacks on vulnerabilities in the Elasticsearch service.

2.2 Logging

Logging automatically registers events in an operating system, covering different levels, actions of users, conditions, transactions or some other information, which is later written into a log file. Although there is not a unified logging standard, logs typically contain additional information, including time stamp, category and description.

Systematic collection of logs is critical for identifying security incidents, providing information about unusual conditions, and occurring problems and failures both on the hardware and software levels.

2.2.1 Syslog standard

Syslog is a common logging standard [20], described in RFC 5424. Conceptually, it distinguishes identities that generate logs, forwards and gathers them, and is designed to transport log messages from a generator to collector, utilising the User Datagram Protocol (UDP). Log messages are often labelled with a facility code and a severity label.

The facility code indicates the type of the software that generated a log. There are 23 facilities, and some of them are presented in Table 2.1 [16]. Facilities codes from 16 to 23 are reserved for local use.

Table 2.1. Facility code

Numerical Code	0	1	2	3	4
Facility	kernel	user	mail	system daemons	security

A severity label represents the priority of a log message. It takes values from 0 to 7, for instance, including such severities as “emergency”, “error”, “informational” and “debug”.

2.2.2 Rsyslog

Rsyslog (the rocket-fast system for log processing [2]) is an open-source utility implementing the basic syslog protocol, and it is used on most Linux distributions as the default syslog daemon. It has content-based filtering, usage of TCP and provides flexible configuration options. It supports a high number of messages per second, direct logging into databases, security add-ons and data compression during transactions [13]. Rsyslog is configured by putting rules, modules and directives in the `/etc/rsyslog.conf` file. A filter in a rule specifies a subset of log messages and an action that should be applied to this subset.

2.2.3 Logrotate

Logrotate is a system utility that automatically rotates and compresses files with logs. A log rotation denotes the process of renaming the current log file by adding a sequence number or date and creating a new log file instance. The old log file also can be compressed, moved to an archive directory, deleted or mailed. This process is used to prevent the storing of extremely large-sized files and outdated log information.

Logrorate has extensive configuration options, including such parameters as a periodicity of the log rotation, its conditions (for instance, the file size), the number of log files to store, and performing a compression. These settings are defined in the `/etc/logrotate.conf` configuration file.

2.2.4 Docker logging driver

Docker logging drivers are multiple logging mechanisms that provide information from running containers and services. Currently, Docker offers 11 logging drivers [1], including the default one and `none` when no logs from containers are available. For instance, some other logging drivers are `syslog` that routes logs to a syslog server, `journald`, `fluentd` and `splunk`. The default Docker logging driver is `json-file`.

2.3 Cluster analysis

Data clustering is the task of splitting a given set of objects into disjoint subsets, called clusters, so that each cluster consists of similar objects, and the objects of different clusters differ significantly. The clustering algorithm is a function: $X \longrightarrow Y$, which assigns a label of the cluster $y \in Y$ to any object $x \in X$. The task

of clustering belongs to a wide class of unsupervised learning tasks. Clustering (unsupervised learning) differs from classification (supervised learning) by the fact that the labels of cluster objects are not initially set, and even the set of clusters itself may be unknown [37]. The task of clustering requires classifying of objects only on the basis of their similarity with each other. The initial data is the matrix of pairwise distances between objects. Cluster analysis is applied in different fields, including machine learning, data mining, pattern recognition, image analysis, text mining and web cluster engines [8].

Clustering tasks can target different objectives, for instance:

- identification of a cluster structure for a better understanding of data;
- data compression; when only the most typical representative from each cluster is left, removing other samples to reduce the original data;
- novelty detection; detection of atypical objects that can not be assigned to any of the previuosly existing clusters [7].

The solution of the clustering problem is fundamentally ambiguous. Different quality criteria and heuristic algorithms lead to different clusterings. There is no universal clustering algorithm, and each heuristic suits only a certain class of tasks. Thus, clustering is an iterative process of knowledge discovery rather than an automatic task.

2.3.1 Cluster analysis using Python

The Python language is widely used for computer science [28], and it includes such libraries as NumPy, Pandas, SciPy, Scikit-learn and Matplotlib. Moreover, the IPython command shell provides interactive computing, including convenient visualisation of data and high-performance tools for parallel computing.

SciPy is a Python library for scientific and technical computing. It provides a set of common numerical operations on top of the data structure of a numeric array, realised by NumPy [25]. It contains a wide range of modules for common tasks in science and engineering, including modules for linear algebra, integration, optimisation and image processing. Apart from other sub-packages, it includes the `cluster` package for performing clustering analysis. It implements such types of clustering as hierarchical clustering, K-means and vector quantization.

3. System model and requirements

This chapter defines the system model and requirements for the monitoring and analytics system of scalable honeypots.

3.1 System model

In the simple approach, logs from honeypots are gathered as presented in Fig. 3.1.

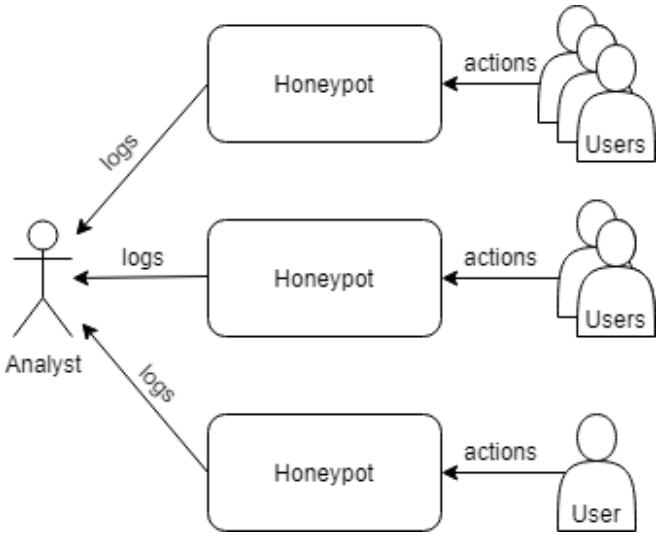

Figure 3.1. Component schema.

This default system model when using multiple honeypots is not scalable, as explained in Chapter 1, and an analyst is forced to work with a large amount of unstructured data. The default honeypot system can be improved by introducing the monitoring and analytics subsystem that supports scalability.

The component schema of the proposed monitoring and analytics is shown in Fig. 3.2. The monitoring and analytics system exists to support the work of an analyst. The Analyst interacts with the system, receiving prepared analytics data that is used to make decisions based on actions of honeypots users. The term "user" is used to refer to anyone who uses the functionality provided by the

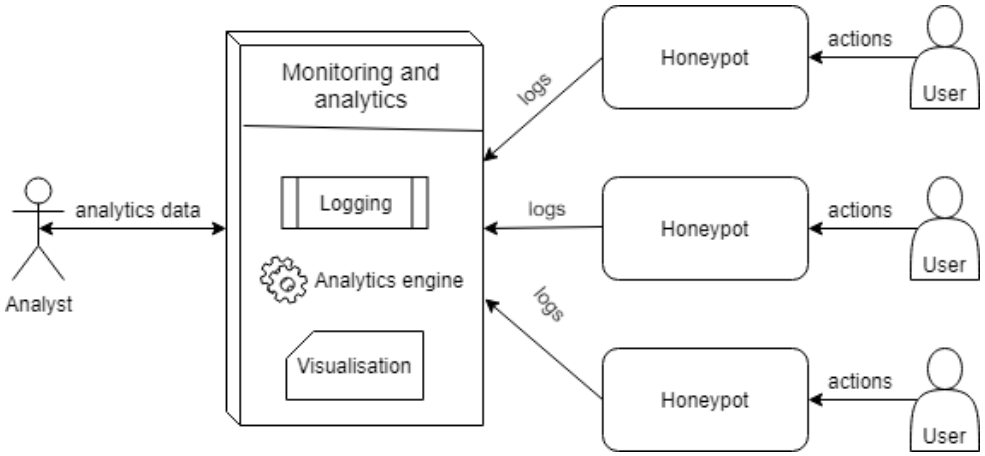

Figure 3.2. Component schema.

honeypot (i.e. usually an attacker).

Actions of users are recorded as logs, which are collected together via a logging system, and represent data that should be analysed.

The monitoring and analytics system itself consists of three components:

Logging is the first component, which centrally collect logs from the whole range of honeypots. Gathered logs contain information about the behaviour of users interacting with these honeypots and their attempts to find and exploit vulnerabilities.

An analytics engine is the second component. It reads logs collected by the first component and separately classifies them into groups connected to certain types of vulnerabilities. Conducting the analysis of this data, the analytics engine becomes able to identify patterns of behaviour of users. The resulting data is used by the analyst for making decisions.

A visualisation of data received from an analytics engine is the third component. The work of an analyst requires interaction with analytics data. Visualisation provides a convenient way to present it to an analyst, showing structured information about patterns of behaviour.

3.2 Realisation of system model

The monitoring and analytics system for honeypots may serve several purposes. First, it can be used in organisations with honeypots that detect hackers, and record and analyse their actions. In that case, the analytics data allows the analyst to learn targeted features of the resource, which then can be defended with a greater accuracy in the real system.

Second, the monitoring and analytics system for honeypots can be used for study purposes in, for instance, a university course of Information Security. Students would interact with honeypots as a task to learn more about vulnerabilities and ways to find them. In that case, data from the monitoring and analytics system can be used to provide information to course staff about the most common approaches of the students, trying to find weak places of a honeypot. Moreover, it can serve as a source of feedback to students about their actions.

3.3 Requirements

The requirements are a foundation for a system design, its future evaluation and making a decision whether the implementation has succeeded. Several categories of requirements have been derived, answering RQ1.

3.3.1 Functional requirements

1.a. Completeness Log data must be collected from all containers created in the system.

1.b. Structured representation Logs must be shown in a scalable way.

1.c. Analytics Analysis engine should identify patterns of behaviour of honeypot users.

These requirements are essential in supporting the analysis of honeypot user behaviour. In case of loss of logs from some containers, less information for analysis will be gained, and correctness of identification of patterns would decrease. Information coming from honeypots should be easy to use by analysts so that logs of a certain user and for a certain kind of vulnerability can be shown. Identification of patterns is aimed to support the work of an analyst to understand the behaviour of users and to provide information on which vulnerabilities are the most targeted ones.

3.3.2 Performance requirements

2.a. Scalability The number of honeypots must not be limited due to performance bottlenecks in the logging system.

2.b. Reasonable performance Logging system should not significantly slow down

the running honeypots.

As the monitoring and analytics subsystem is built over the honeypot functionality, it should not produce a negative effect on it. The case study system that runs honeypots does not limit the number of containers it can start. This is important for measuring the activity of users and should not be diminished by the implementation of a monitoring subsystem. The slowdown of the system will lead to a decrease in the usability for users and should be avoided.

3.3.3 Deployability requirements

3.a. Low resource cost The monitoring and analytics system should not significantly increase the resource usage of the overall system.

3.b. Usage of open-source software The monitoring and analytics system should use only open-source software.

The usage of additional resources and commercial software incur costs that can be avoided by resorting to free software. Furthermore, if more resources are required for the monitoring and analytics subsystem, fewer resources for running honeypots would be available, which may lead to violation of requirement 2.a. The wide range of free software in appropriate areas corresponds to the set goals at a satisfactory level, providing all required functionality.

4. Case Study

The existing system with honeypots from the Information Security course at Aalto University was chosen for the implementation of monitoring and analysis. It matches with the system model and has the requirement for scalability to support at least 200 students in the class.

This system allows students to experiment with concepts learned in the Information Security field. They learn which types of vulnerabilities exist and how to find and exploit them. At the same time, there is no risk that students will crash any real system or interfere with each other's work.

The used honeypot system is neither a production not research type of honeypot. It is not a production type, as the main purpose of the system is not to defend the internal network. The system may be connected with research type, however it does not research the action of attackers, rather providing students the ability to learn certain types of vulnerabilities through practical experience and examines their actions inside honeypots. Thus, this type of honeypot can be categorised as *educational*.

Education honeypot systems have a large number of attackers, allowing collection of lots of data in a short time. Therefore, they are beneficial in developing monitoring techniques and data analysis. Although education honeypot systems are not intended for detection of new attacks and vulnerabilities, they can help to understand the approaches taken by attackers to find common vulnerabilities. These systems assist in exploration how the attackers learn and how their malicious behaviour evolves.

4.1 Exercises

There are currently three exercises on the Launcher to exploit the vulnerabilities described above:

The first exercise is about SQL Injection and provides a web-application honeypot with the corresponding vulnerability.

The second exercise consists of bypassing of client-side validation, buffer overflow and server-side poisoning. This exercise on the Launcher runs two honeypots: a web-application honeypot with the enabled server-side poisoning vulnerability and an IoT device honeypot.

The third exercise is intended for XSS, both stored and reflected. It also runs a web-application honeypot.

Therefore, during the Information Security course a user interacts with four honeypot instances: three instances of the web-application honeypot and one IoT device honeypot.

4.2 System overview

The scheme of the existing honeypot system is presented in Fig. 4.1.

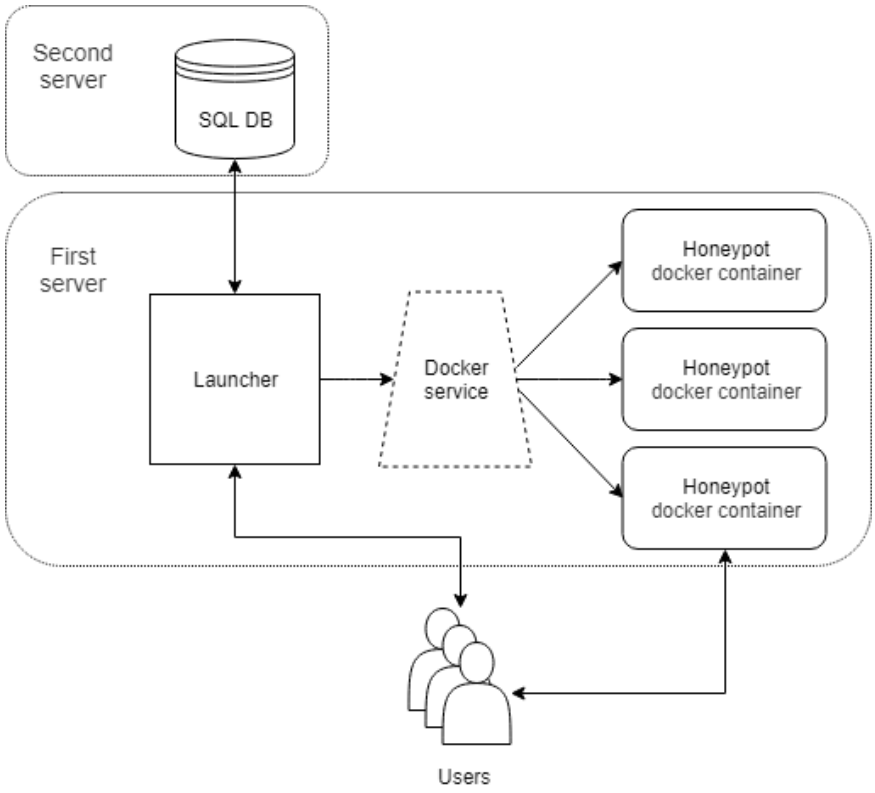

Figure 4.1. Schema of the honeypot system with exercises.

First, users reach the launcher by a link. The launcher is presented as a web-site that contains all types of exercises for the course. It is a Flask application running on Apache2 on the first sever. Then, using the launcher, users choose an exercise to

complete. Then, they are directed to the page where they can run a honeypot of the corresponding type. Honeypots are running on the same server. A new honeypot is created for every user for every exercise; a unique link to it is presented on the launcher. The launcher has *submit* buttons for every exercise, which are clicked by the student after finishing an exercise task. When solutions are checked, the corresponding number of points goes to the MySQL DB with marks of students, which is located on the second server.

4.3 Types of honeypots

Two types of honeypots exist in the system: web application and Internet of Things (IoT) device. The web application honeypot presents a high-interaction type of honeypot, providing a real operating system with fully functional services that can be compromised by a user. The IoT device honeypot is related to a medium-interaction type of honeypot that only emulates services, however providing fake responses for requests of users.

The honeypots are implemented as docker containers, chosen for their ability to require small amount of resources, so a relativity large number of lightweight containers can exist in the system at the same time.

The Docker service is running on the same server with the launcher. It contains images for the types of honeypots. Every time the user runs a new honeypot, a new container is created, based on the corresponding image.

4.3.1 Web application

The web application docker container uses Node.js and sqlite3. It is emulating a website with information about pot plants of different people. This honeypot provides several types of vulnerabilities, characteristic of common web-sites: SQL injection, bypass of client-side validation, server-side poisoning, stored and reflected cross-site scripting (XSS).

The search field in the application is intentionally made to be vulnerable to a SQL injection by inclusion of a raw query in a SQL request, Listing 4.1.

Listing 4.1. SQL injection vulnerability.

```
var sql_query = 'SELECT name, color, planttype, potsize,
   shared FROM plants WHERE user_id=' + req.user.get('id') +
   ' AND name LIKE "%' + plant + '%";';
```

The example of a client-side validation that can be bypassed by users is the usage of javascript check of user input in the HTML code of the sign up page,

shown in Listing 4.2, which is not rechecked on the server side.

Listing 4.2. Client-side validation.

```
$(document).ready(function(){
    $("#signup").submit(function(event) {
        var username = $('#username').val();
        if (username.length > 20) {
            $('#prevalidation_error').show();
            $('#prevalidation_error').html('Username cannot be
                longer than 20 characters');
            event.preventDefault();
         }
     });
});
```

The server-side poisoning vulnerability is implemented by using a dangerous eval() function in sorting, which executes a string parameter that can be changed by the user, Listing 4.3.

Listing 4.3. Usage of eval() function.

```
var field = req.body.sort_field;

if (config.code_injection == 'enabled') {
    retPlants.sort(function(a, b) {
        var av, bv;
        with (a) {
            av = eval(field);
        };
        with (b) {
            bv = eval(field);
        };
        return av >= bv;
    });
}
```

In addition, the server is vulnerable to a stored XSS, as it does not execute filtering of input of users that is stored on the server. The reflected XSS is also enabled by the code for displaying an error page, which is decoding URI, Listing 4.4.

Listing 4.4. Vulnerability for reflected XSS.

```
$(document).ready(function(){
    $('.container').html("Oops. Page " + decodeURI(window.
        location.pathname + window.location.search) + " could
        not be found :-(");
});
```

Users are encouraged to exploit these vulnerabilities by executing the tasks, following the instructions they receive during the course. The goal of the task may be formulated as: “To submit your solution to this part, copy one plant name from the other users to the Exercise Launcher” or “Your goal is to register with a username that is longer than the limit” [12].

4.3.2 IoT device

The IoT device is simulated by a server written in C and executed on the docker container running Ubuntu. It listens on a TCP port and receives commands from users, which includes authentication code (HMAC-SHA256). The IoT device honeypot contains a buffer overflow vulnerability, which is executed by sending specially constructed command to a device. The HMAC input array is defined after the keyphrase variable in the code, which is shown in Listing 4.5 [12]. Therefore, the right length of the command overwrites the secret keyphrase. Thus, the goal of the users is to make the device to execute an unauthorised command by exploiting this vulnerability.

Listing 4.5. Buffer overflow vulnerability.

```
void handle_message(int connectfd, char *msg) {
    char *command_rec;
    char *user_rec;
    char *hmac_rec;
    unsigned char *hmac;
    char keyphrase[MAXKEYLEN+1] = {0};
    char hmac_input[MAXHMACINPUTLEN+1] = {0};
    char hex_hmac[HMACLEN*2+1] = {0};
    char reply[MAXREPLYLEN+1] = {0};
    int keylen;
    int i;
    ...
}
```

4.4 Limitations of the system

As all tasks are checked automatically, the staff of the course do not need to interact with honeypots, it is enough to check the number of points in the database. However, this only indicates whether the task was successfully completed, but not how it was done. This approach limits the ability of the staff to analyse how students were executing tasks, which and how many attempts they have done and what are common failed approaches to exploit vulnerabilities among students.

The implementation of the monitoring and analytics system integrated with this existing honeypots system will remove these boundaries, collecting all necessary data and providing some analysis to make the work of the personnel easier and provide a better visualisation of data. With that new functionality students also would benefit as the staff would have the possibility to react on their actions and provide some additional information.

5. Monitoring system design and implementation

The monitoring and analytics system for scalable honeypots was built based on the system model described in Chapter 3 to meet the requirements presented in that chapter. The honeypot system introduced in the case study (Chapter 4) was taken as a foundation. The description of the system is divided into monitoring, analysis and visualisation parts.

The monitoring system is responsible for the collection of logs from honeypots, their aggregation and deriving logs of malicious actions. It allows exploration of the behaviour of users during their interaction with honeypots. It serves as a foundation for the analysis subsystem, containing all attempts of users to exploit vulnerabilities in one place, prepared for further analysis.

The monitoring system is described in this section, answering RQ2. An overview of the monitoring system is shown in Fig. 5.1.

5.1 Logging inside application

To prevent honeypots from being black boxes for an analysist, it is important to put logging inside them. As we consider honeypots with known vulnerabilities, it is meaningful to log the exact actions that are trying to exploit these known vulnerabilities. That would reduce the amount of stored data and provide accurate structured information for the analysis.

5.1.1 Web application

Inside the web application, all possible input fields with vulnerabilities are logged, covering SQL injections, server-side poisoning and stored XSS. This logging considers all potentially sensitive forms, including situations when data can be inserted in inappropriate fields, for instance, changing of default values in options for sorting results of a query. Thus, all attempts of users to exploit these vulnerabilities are collected.

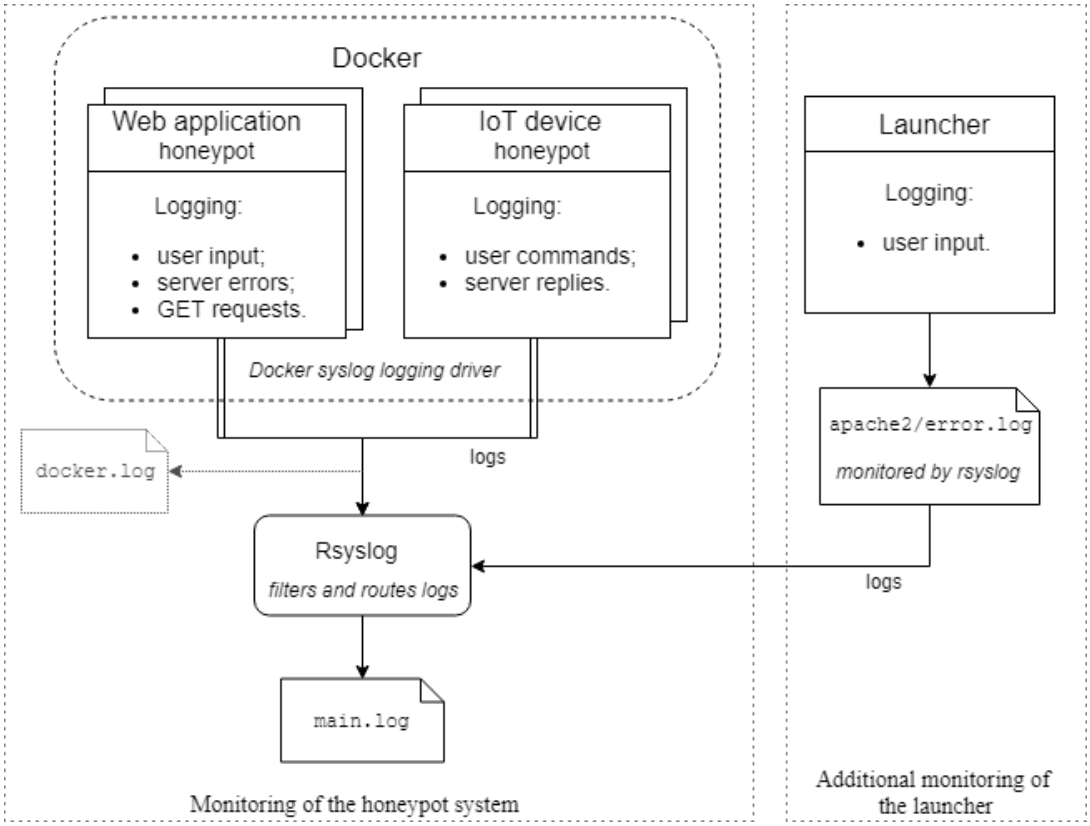

Figure 5.1. Scheme of the monitoring system.

An example of logging of an input field with SQL injection vulnerability is presented in Listing 5.1.

Listing 5.1. Logging of the input field.

```
router.post('/search', function(req, res) {
    var plant = req.body.plant;
    var successCallback = function(plants) {
        if (plant!="") {
            console.log('searchPlantsInput:', plant);
        }
    ...
    }
...
}
```

Among with data coming from users, it is useful to log errors appearing inside a honeypot for a better understanding of processes on a server. In addition, this information can be used later in the analysis indicating what types of errors are typical for attempts of users to exploit vulnerabilities.

SQL errors and the queries which generated them are logged inside the honeypots. Therefore, logging of this field is done only when the SQL query is successful

to prevent redundancy. This case covers both situations when the user has succeeded with the search and when the SQL injection was constructed incorrectly, so the SQL parser has recognised it as a string. In addition to SQL errors, errors appearing on a server during attempts to perform server-side poisoning are logged.

Bypass of HTML code conditions is done on the client side, so the success of that attack is checked by the honeypot system, however, the exact details cannot be recorded by server-side logging.

To check a reflected XSS, users leave the malicious link that steals cookies in a special field on the launcher (not a honeypot), which is also logged. This additional logging is not needed for general honeypot systems where users interact only with a honeypot. Meanwhile, the considered system provides the launcher web-site for students, and the further analysis will benefit from adding some extra logging on it.

5.1.2 IoT device

All commands sent to the IoT device honeypot are printed into the standard error (stderr) stream. Also we can log responses from the emulated IoT device, which contains information whether the command from a user was accepted and the reason of failure if not, to use it later for the analysis.

Thus, we have a collection of logs from the launcher and from each instance of two types of honeypots. These logs should be gathered together in one place on the server that runs them.

5.2 Logging in Docker

To receive information from running honeypots, the docker logging mechanism, called *logging driver*, is used. The default logging driver stores logs in the json-file, which can be viewed by `docker logs` command. Instead of that, syslog logging driver was chosen, sending logs to a rsyslog server, which provides more possibilities for routing and filtering logs. Rsyslog implements the basic syslog protocol with rich capabilities and is already preinstalled in a variety of Unix systems, including Ubuntu.

The configuration file for running honeypots on the launcher was enlarged with information about the selected type of the logging driver and the syslog tag. The syslog tag was chosen to consist of the daemon name and a container name, which includes a student number and a name of the exercise. The information from the configuration file is used as arguments while starting a new container; it also

included such information as, name of the image and the port. Therefore, all exercise containers will run with the chosen logging driver and tag.

5.3 Gathering of logs

The configuration of rsyslog was changed to route all logs from running honeypots (both web application and IoT device) to a `docker.log` file. That was done by using the daemon name from the syslog tag as a filter.

The monitoring and analytics system does not require all logs generated by the honeypots. As the system provides customised logging that contains only malicious actions, we do not need to create or store the file with all default logs (`docker.log`). It was left in the system for debugging purposes, but can be removed later. However, the honeypots systems that are intended to explore previously unknown types of vulnerabilities would need this file for the analyses of new attacks. Such versions of systems are considered later in the Discussion Chapter 8.

All customised logs were done in a way to contain a *unique string* for filtering purposes. This refers to logs inside web application including errors, logs of commands to the IoT device and its replies, and logs from the launcher. The string "searchPlantsInput:", for instance, is used for one user's input inside the web application; it can be seen in shown earlier Listing 5.1.

A small number of logs already contained a string by which they could be easily identified and filtered, for instance, "SQLITE_ERROR" in SQL errors logs. Therefore, such logs did not require manual adding of extra information in them.

Thus, all logs with such unique string sequences are derived by rsyslog and put in a separate `main.log` file. This file is supposed to consist only of malicious actions of users, as all users of honeypots are considered to be intruders.

Moreover, rsyslog was configured to monitor logs of the launcher in an `apache2/error.log` file to derive users input with the link for stealing cookies. These logs are also transmitted to the `main.log` file.

In addition, logs from docker containers of web application contain all GET requests of users. To cover reflected XSS with a higher efficiency, along with logging the special input for checking the malicious link, GET requests containing "script" sequence are also derived and routed to the `main.log` file.

Thus, all logs about malicious actions of users in honeypots are collecting in one place, allowing an analyst to review them or use later in the automated analysis.

5.4 Log rotation

As the proposed monitoring and analytics system supports scalability, it should be able to operate on significant amounts of log traffic from many honeypot instances. The monitoring system already has all logs from honeypots gathered in one place, as described in the previous section, however it should also take into account the maximum size of the log file.

The issue of large-sized log files is solving by *log rotation*, when current log files are automatically archived and replaced by new ones. The periodicity of log rotation should be chosen according to the amount of traffic produced by honeypots.

In the considered case study, the `logrotate` utility was configured to rotate logs every week, as during the Information Security course there is a large number of honeypot users, but not enough to make the rotation daily. The `logrotate` stores files for 3 months to cover the duration of the course. After that period, log files are not deleted, but archived in a separate directory, so that information from them can be used later for analysis or statistics purposes. Rotated log files are marked with the current date to facilitate the file navigation.

6. Analysis and visualisation

The analytics system is purposed to support the work of an analyst. It obtains data collected by the monitoring system and conducts an automatic analysis on it. This analysis groups the logs into clusters in order to help an analyst identify patterns of user behaviour. The clusters and logs statistics are visualised to demonstrate the information in a convenient way to an analyst.

6.1 Analysis

The analytics system is described in this section, answering RQ3. The scheme of the analytics system is presented in Fig. 6.1.

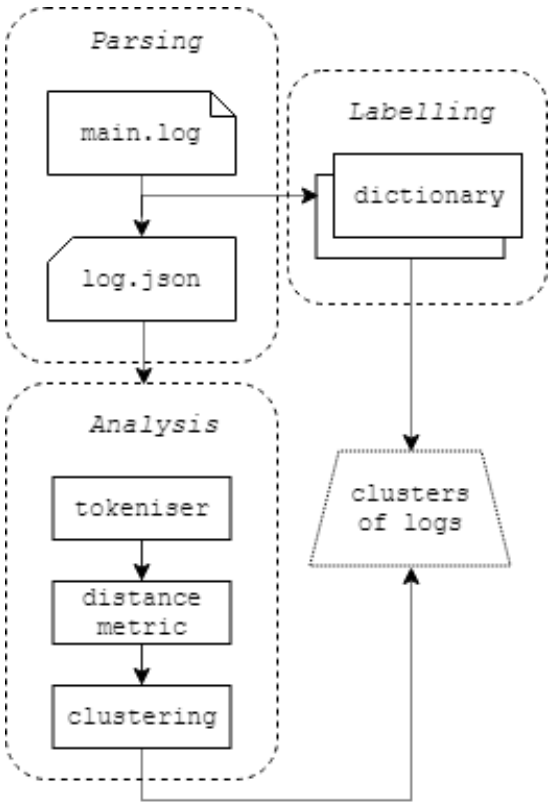

Figure 6.1. The analytics system.

Currently, we consider the education honeypot system with a known set of vulnerabilities that users are intended to exploit. In addition, the types of attacks are divided into different exercises. These features and the implemented customised logging allow us to analyse log entries more precisely, skipping the step of initial grouping of logs by the type of the attack. Other versions of the proposed system are discussed in Section 8.1

6.1.1 Preparation of data

To prepare logs for the analysis, information from the `main.log` file should be parsed, removing extra information and aggregating logs by types of the attack.

Parsing of logs is executed by using regular expressions in Python. The type of the vulnerability that users have tried to exploit can be derived by parsing the syslog tag, introduced in Section 5.2, as it contains the name of the exercise, and the unique string consequence in a log. Number of the student is also taken from the tag. The example of a raw log is presented in Fig. 6.2.

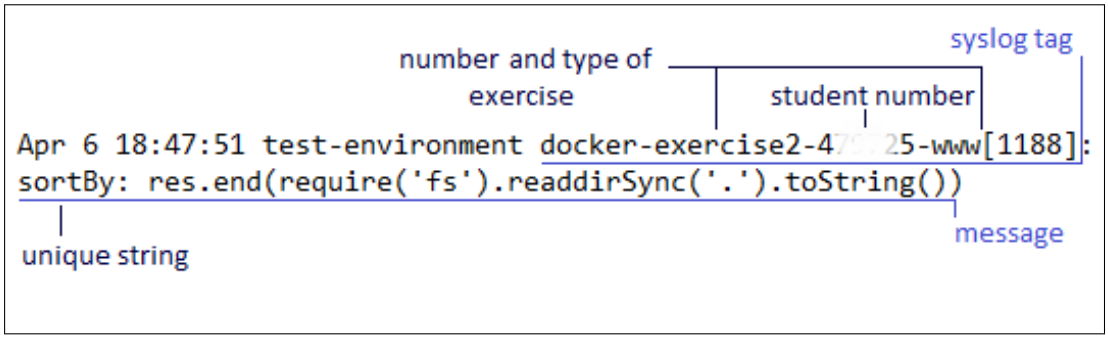

Figure 6.2. Single log from the main.log file.

The derived types of the attacks from the existing in the honeypot system exercises are: “SQL injection”, “Buffer overrun”, “Stored XSS”, “Reflected XSS” and “Server-side poisoning”.

Along with logs of malicious activity of users, the monitoring system collects logs of three types of errors occurring inside honeypots: SQL errors and errors from the server during server-side poisoning – from the web application, and errors emulated by fake responses of the IoT device.

To separate logs that have caused errors from the general ones, two extra types of logs were added: “SQLi with errors” and “Poisoning with errors”.

“SQLi with errors” means logs with SQL injections received with SQL error. These SQL errors can serve as ready labels for groups of SQL injections, so they should not be analysed with general logs of SQL injections without labels and are distinguished as a separate type. However, they still can be combined together

to demonstrate the statistics of this attack. The same applies to “Poisoning with errors” and “Server-side poisoning” types of logs.

All commands from users to the IoT device receive informational responses, therefore there is no need to separate them into two groups of logs.

To store parsed data, the JSON format was chosen, because of its ability to store key-value and set of values information that suits received data. In addition, it can conveniently be converted into a Python dictionary for further analysis and data visualisation.

After the parsing, data is stored in `logs.json` file and is presented in the following format: `{student number : {type of the attack: [log, ...], ...}, ...}`. In this format logs can be accessed by the key, and this file can be used to explore the statistics of actions of the users.

6.1.2 Labelling

As was mentioned in the previous subsection, there are 3 types of logs that already have labels by which they can be grouped: “SQLi with errors”, “Poisoning with errors” and “Buffer overrun”.

To store errors and the corresponding user actions, three new dictionaries are created during the parsing of the `main.log` file along with the common dictionary for logs. These dictionaries have the following format: `{ type of error: [log, ...], ...}`. Thus, all logs of the same type with the same label can be easily accessed by the key.

These dictionaries present logs that do not require a further machine learning analysis, as they are already grouped by the same type of error. This classification of logs supports the work of an analyst showing common patterns of users behaviour, specifically when users attempt to exploit a certain type of vulnerability, with an informational label, explaining why they were grouped together.

6.1.3 Machine learning analysis

Besides the logs described above, there are logs that do not have any context. They are “SQL injection” and “Server-side poisoning” types of logs that have not caused any error, and “Stored XSS” and “Reflected XSS” logs that always come without any additional information.

Thus, if these logs stay in the initial state without an additional handling, they will be represented as unstructured data, and will be difficult to analyse. To support an analyst in processing this type of data, the proposed system uses machine learning techniques for data analysis.

In particular, cluster analysis suits this task as it allows grouping of data in unsupervised manner without knowledge of cluster labels. Therefore, the applying of clustering to unlabelled logs in the system will provide an automated analysis, enabling an analyst to skip manual aggregation of logs. Viewing clusters of logs of the same type will assist in identifying of patterns of users behaviour inside honeypots.

The process of the analysis can be divided into tokenisation, usage of metric and clusterisation.

After parsing logs from the system and their preparation in the previous steps, the execution of the analysis can be started. For that, the code aggregates the logs of different users together by the type of the attack, handling the Python dictionary received from the `logs.json` file. As was mentioned previously, labelled logs are not analysed by machine learning techniques.

Tokenisation

As string characters are analysed, a *tokeniser* can be applied to them. To make tokens from a string, it can be split on words. In the case of the honeypot, in this step we can get rid of uninformative tokens to make the analysis more efficient.

Here is the example how logs with SQL injections are tokenised; first, we split a log on words and non alphabetic characters. Then, all words are checked whether they are SQL command words, if not, they are removed from the tokens. Thus, tokens consists only of non alphabetic characters and SQL command words. This step removes all data that is not meaningful for the analysis and should not influence the clustering. For instance, when the analytics system is used to analyse the structure of SQL injection attacks, the names of columns can be ignored. On the other hand, if the analyst is trying to explore which data is targeted by the attackers, the SQL commands can be ignored instead.

The example of the log with SQL injection before and after tokenisation is presented in Fig. 6.3.

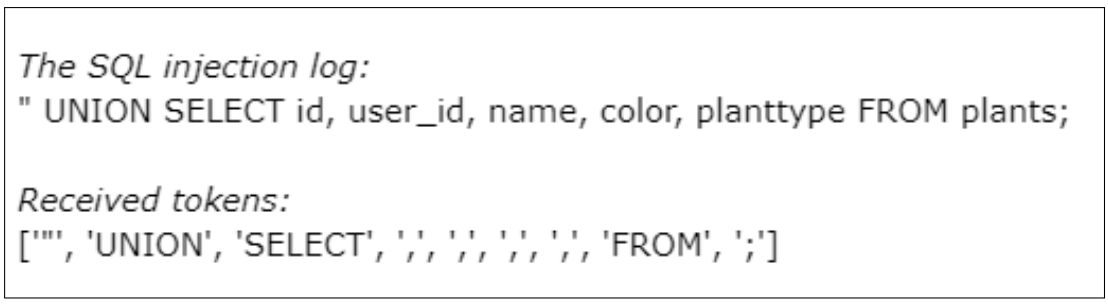

Figure 6.3. Tokens for SQL injection.

For other types of the attacks other commands words can be chosen for filtering,

JavaScript or HTML, for instance. However, tokenisation of a log string is not required for the further processing, it can be treated as a single unit. To a greater extent this applies to the cases when in a honeypot system there are honeypots without a known set of vulnerabilities or honeypots that aim to detect a new types of attacks. In addition, it applies to the cases when a logged field in a honeypot is vulnerable for several types of attacks.

Metric

Before performing the clustering, a string metric is applied to the logs to measure the distance between them. For that purpose, the *Levenshtein distance* [22] is used that refers to the edit distance class of string metrics family.

It allows measuarement of the distance between two strings by counting how many iterations is needed to transform one string to another by using such operations as the removal, insertion, or substitution of a character in the string [42].

In Python, the Levenshtein distance function can take both a whole string and an array of tokens. In the first case, it measures how many characters in a string should be edited, while in the second case, it measures how many tokens in an array should be changed, but not characters inside a token.

The Levenshtein distance is applied to arrays of logs that have already been tokenised or left as strings. The distance between every two logs is measured, thus creating a $n \times n$ matrix, where n is the number of logs.

Clustering

Unsupervised machine learning allows representation of a hidden structure of the data that does not have labels [23]. We use clustering algorithms to find groups of similar logs, based on distances calculated in the previous step.

One of the good approaches to perform clustering on the text data is the usage of the *agglomerative hierarchical clustering* [3], which can be found in SciPy library. It builds a hierarchy of clusters with a predetermined ordering from bottom to top (every log is a separate cluster that later is recursively merged with the closest one).

The usage of the hierarchical clustering does not require a priori knowledge of the number of clusters. It is a valuable characteristic as the number of clusters is undefined in the considered honeypot system.

To decide when to stop merging, the determination of the maximum permitted distance between logs in a cluster is required. That can be done by obtaining a *dendrogram* of the hierarchical clustering. The dendrogram is a tree diagram, built on a matrix of proximity measures, frequently used to show the arrangement of the clusters. The example of a dendrogram is presented in Fig. 6.4 where

distances are presented along the Y-axis.

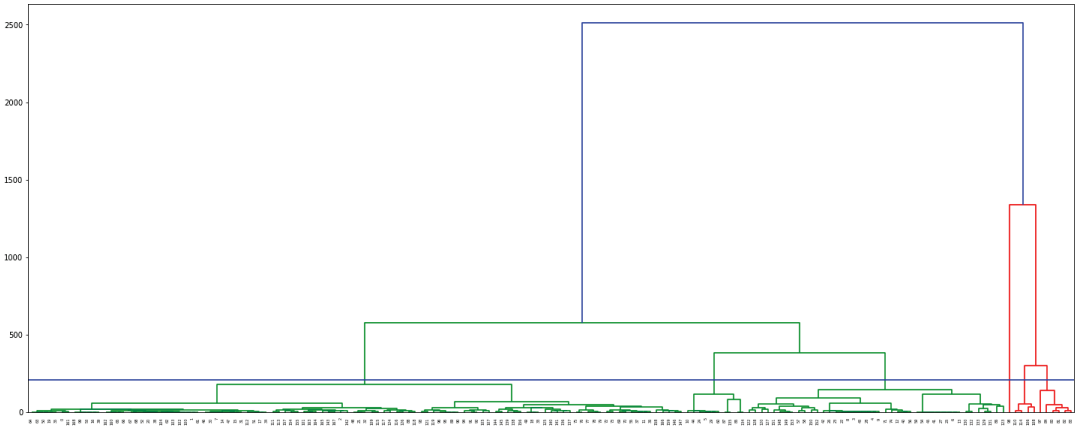

Figure 6.4. The dendrogram of hierarchical clustering. The x-axis represents indices of samples, and the y-axis representes distance between clusters.

By looking at a dendrogram, the maximum distance can be defined visually, as cutting a dendrogram at a certain level provides a set of clusters. The example cutting of a dendrogram can be seen in Fig. 6.4, where the maximum distance is presented by the blue horizontal line.

However, this parameter can not be defined once, it depends on an analysed data and should be corrected manually. The maximum distance may be adjusted after receiving the clusters if required accuracy was not achieved. For instance, the maximum distance can be enlarged if some logs that represent the similar behaviour of users were divided into separate clusters.

Along with the visual method, special coefficients measuring the accuracy of clustering can be used to define the maximum distance. *The Silhouette Coefficient* [6] for a sample is calculated by $\frac{b-a}{max(a,b)}$ equation, where a is a mean distance between a sample and other samples in a cluster, and b is a distance between a sample and samples from the nearest cluster.

Clustering returns the result as a set of numerical labels. For the convenient storing and handling of the results, a dictionary is created with a label as a key and an array of logs of the same cluster as a value.

Thus, along with labelled clusters, an analyst has all logs that have not caused any errors inside honeypots clustered as well. Instead of viewing a long list of raw logs and trying to group them manually, an analyst can concentrate on working with prepared clusters that represent similar attempts to execute an attack in a honeypot system. This assists in identifying the patterns of users behaviour and, when considering an Information Security course, common failed approaches of students to exploit vulnerabilities.

6.2 Visualisation

The monitoring and analytics system handles a significant amount of data produced by honeypot instances. To present this data in a form convenient for an analyst, it should be visualised.

For that purpose, the dashboard-type web-site was chosen. It provides an acceptable level of interaction and the ease of use at the same time. The data that is presented on the web-site can be seen in Fig. 6.5.

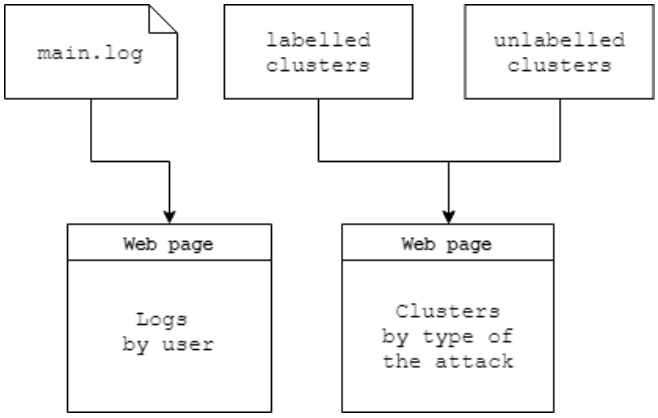

Figure 6.5. The scheme of data visualisation.

Visualisation of data is depends on the honeypot system we consider. Currently, in the case study system there are two web-pages for demonstrating:

- logs of a specific user, as it is important to show attempts of a student to the stuff of the course;

- received clusters of logs by the type of attack, to be able to analyse the common failed approaches of students to exploit vulnerabilities.

For other honeypot systems that do not differentiate users of honeypots, the first page can be replaced by the statistics page of general number of attacks by their type or some other information, which may be useful for that type of the system.

The mentioned web-pages are implemented on the same site that contain the Launcher of the exercises.

The web page with logs of students contains a field for the choice of logs by a student number. It demonstrates the chosen logs by the type of the attack. The structure of the web page with logs of students is presented in Fig. 6.6.

The web page with clusters is designed similarly, it has the choice of the type of the attack and demonstrates the received clusters whether labelled (keys from the dictionaries of labelled clusters) or with a numerical labels (“Cluster 1”, for

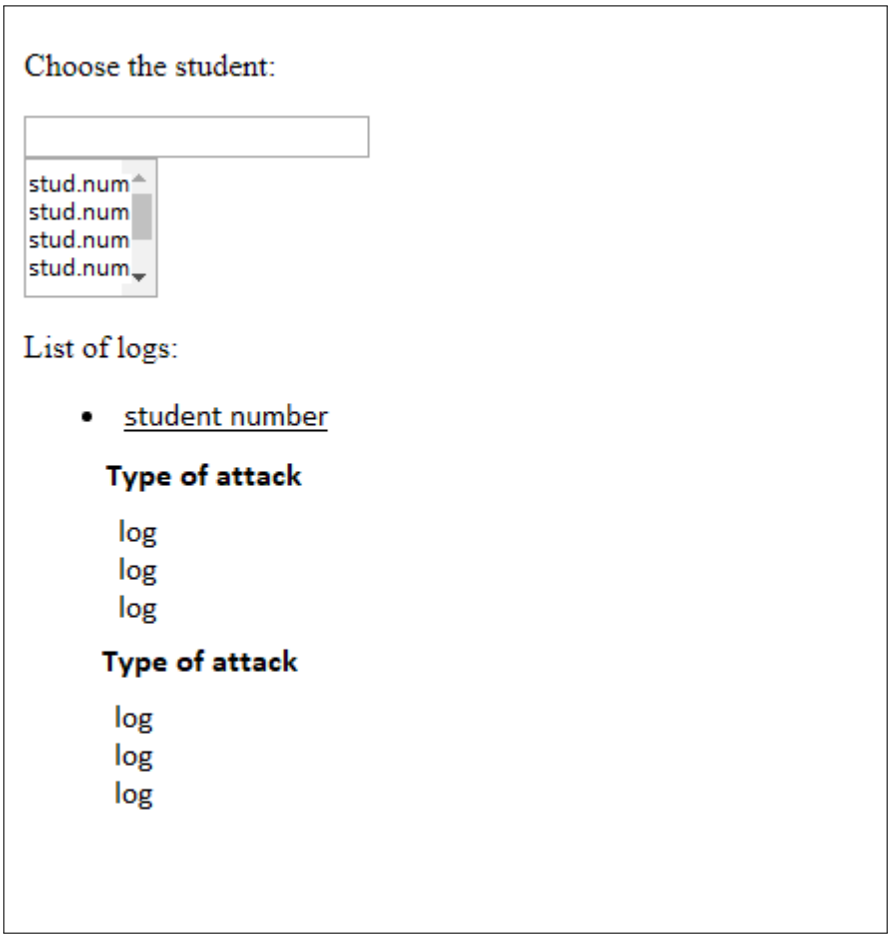

Figure 6.6. The structure of web page with logs of students.

instance) if they were distinguished during the machine learning analysis.

7. Evaluation

This chapter evaluates the implemented monitoring and analytics system, covering the requirements described in Chapter 3. In addition, it presents the results of the conducted user study.

7.1 Pilot user study

The pilot user study was conducted for testing the implemented monitoring and analytics system and checking that the system fulfils the defined requirements. A full user study is planned to be executed in the autumn semester during the Information Security course. Then, the system will have a large number of users to monitor and therefore, more data to analyse, learning attacker behaviour.

7.1.1 Method

To find participants for the user study, we have emailed current computer science students, sending them a form with conditions of participation, including the deadline and the reward (two movie tickets). Seven students who have filled the form have participated in the user study. One of them is currently on the bachelor level of study and others are masters. Six participants have some security background, and only one of them has not previously attended any security courses. The selection criteria of the involvement was the requirement that a participant has not had a previous experience of executing tasks on this platform, so they behave as typical students of the Information Security course.

Each participant received instructions about interaction with the exercise Launcher and running honeypots, and a set of exercises of the Information Security course that they should attempt to execute.

During the user study, the Launcher was set to communicate with a separate database for testing, so the results of participants were not put in the same database with real students of the Information Security course. The previously

received logs were removed from the `main.log` file.

7.1.2 Received data

During the user study, 17,622 log entries in total were received. Among them there are 8,920 logs of errors and 6,444 repeated (by the same user) logs of users actions. These numbers are explained by the fact, that during the execution of the Buffer overflow attack some students were using scripts that were repeatedly sending commands to the IoT device, and it was replying back. The repeated logs do not go to the dictionary of logs to prevent redundancy. The number of logs that contains all unique attempts of every user to execute the tasks is 1,870. The number of these logs per student is presented in Table 7.1. 388 logs that contain only alphabetical characters coming from forms fields were not counted, as they do not represent attempts to execute an attack, and do not go to the dictionary of logs and the further analysis.

Table 7.1. The number of unique log entries of attacks per student

Student	Unique logs with attacks
1	128
2	367
3	28
4	1020
5	58
6	148
7	121

The results of the user study were used to evaluate the fulfillment of functional requirements of the system.

7.2 Functional analysis

The main goal of the project is to provide monitoring and analysis of data produced in a honeypot system with supporting of scalability. The evaluation of the functionality of the proposed system and the accomplishment of the goal are based on requirements presented in Section 3.3.1.

7.2.1 Completeness (1.a.)

In total, during the user study 28 honeypots were run, four honeypots per student (this number is explained in the Section 4.1). To check honeypots run by a user, `docker ps -a` command can be used that shows docker containers in the system. The example of honeypots of one user shown by this command is presented in Table 7.2;

some columns were removed from the table (e. g. “PORTS” and “COMMAND”).

Table 7.2. Docker containers of a user

CONTAINER ID	IMAGE	NAMES
7dbebd0a02dc	mesarpe/exercise_server:v4	exercise1-47**25-www
13154da095b2	mesarpe/exercise_server:v4	exercise3-47**25-www
c1772d79ac99	aura/buffer_server:v1	exercise2-47**25-tcp
f566fe3df0be	mesarpe/exercise_server:v4	exercise2-47**25-www

As the name of a container is included in the syslog tag, it is possible to check whether logs from all honeypots of a user were received. The user study has shown that we have collected logs from all 28 honeypots, which means that the monitoring system covers all created in the honeypot system honeypots.

To check that all of the logs created by a single honeypot are recorded by the implemented monitoring system, the automated test was run. The simulated user was filling the form that has logging inside a web-application honeypot. The amount of logs came from this test was checked in the log file of the system. The results revealed that all 315 logs produced during the test were recorded.

Therefore, the monitoring system fulfills the requirement of the completeness, as it records all logs from all running honeypots.

7.2.2 Structured representation (1.b.)

The system was implemented to store all data structurally, by using three types of dictionaries with different types of keys. During the user study, the first dictionary was filled with all received unique logs (1870). This dictionary allows representation of logs by a user, including structure for a type of the attack.

The web-page on the dashboard web-site displays this data conveniently. It was tested with data from the user study, and its visualisation is provided in Fig. 7.1, showing logs of students (a reduced amount) from the dictionary.

Two other dictionaries demonstrate common users behaviour supporting the structure of clusters, both labelled and numerical, and are visualised on the second web page.

The labelled dictionaries were filled with errors and logs that caused them by the type of the attack. The keys from a labelled dictionary of Buffer overrun logs are presented in Listing 7.1.

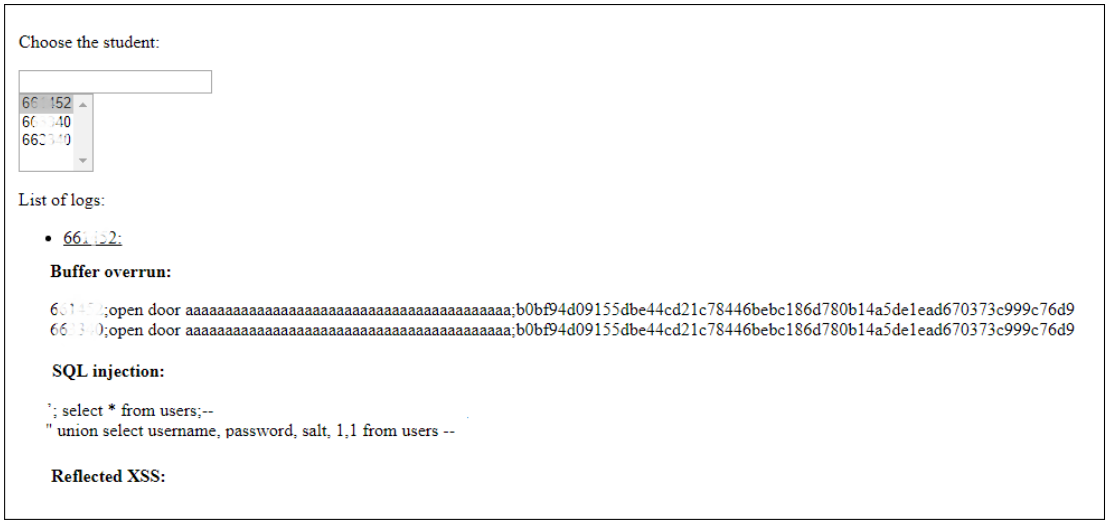

Figure 7.1. The web page with logs of students.

Listing 7.1. Example keys from a labelled dictionary.

```
['Authentication failed. Command not accepted.', 'Invalid
    message. Username must be at least 3 characters.', '
    Invalid message. Hexadecimal HMAC-SHA256 must be 64 hex
    characters.', 'Unknown user.', 'Invalid message. Should be
      user;command;hmac  terminated with newline.']
```

The unlabelled dictionaries were filled with the results of clustering. Some sample logs from the first three clusters of the SQL injection type of attack are presented in Listing 7.2.

Thus, all data in the monitoring and analytics system has a structured representation.

Listing 7.2. Clusters of SQL injections.

```
Cluster 1
;'OR 1=1
OR 1=1
OR 1=1;'
...

Cluster 2
"; SELECT name, color, planttype, potsize, shared FROM plants;
    --
"; SELECT name, color, color, potsize, shared FROM plants; --
'; SELECT name, planttype AS color, planttype, potsize, shared
    FROM plants; --
...

Cluster 3
" UNION SELECT 1,2,3,4,5 FROM users;'#
" UNION SELECT username,salt,3,4,5 FROM users;'#
" UNION SELECT username,salt,password,4,5 FROM users;'#
...
```

The derived clusters allow exploration of learning steps of attackers. For instance, Cluster 2 from Listing 7.1 shows a common failed approach of SQL injection execution when an extra semicolon is inserted in the query. That provides an opportunity to give helpful feedback to students, assisting them in learning vulnerabilities. Moreover, it provides an opportunity to detect attacks before they succeed, as in non-education honeypot systems attackers frequently operate blindly, attempting different approaches.

7.2.3 Analytics (1.c.)

The implemented monitoring and analytics system supports identification of patterns of user behaviour by collecting logs of errors and users actions that caused them, and by processing through machine learning analysis.

The patterns of users behaviour can easily be distinguished from the labelled dictionaries as they contain reasons of grouping expressed in labels with errors. To identify the patterns from other logs, a process of clustering was used.

To demonstrate the results of hierarchical clustering, the array with logs of SQL injections is used. For that type of logs, it can be said that we should have at least

two clusters, as it is known that users will try to perform SQL injections from two different tables following the provided task. However, the resulting number of clusters is unknown, as the system is purposed to find the common patterns of attacks automatically.

The dendrogram for SQL injections logs is presented in Fig. 7.2, showing the hierarchy of clusters.

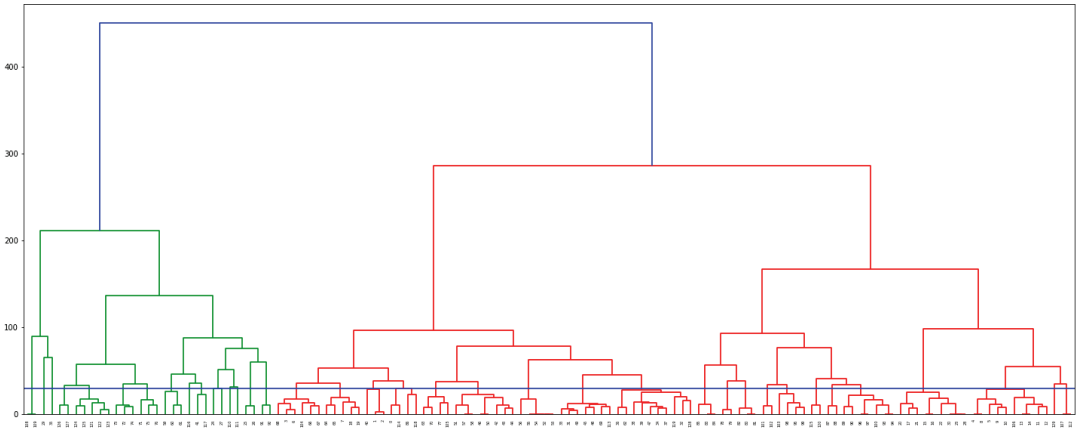

Figure 7.2. The dendrogram of SQL injection logs.

To evaluate the results of the clustering, the silhouette coefficient was used. It measures the quality of clustering in quantitative terms. Therefore, the number of clusters that maximises the silhouette coefficient should be selected.

The silhouette coefficient was calculated for the wide range of distances, the maximum distances and corresponding values of the coefficient are presented in Fig. 7.3.

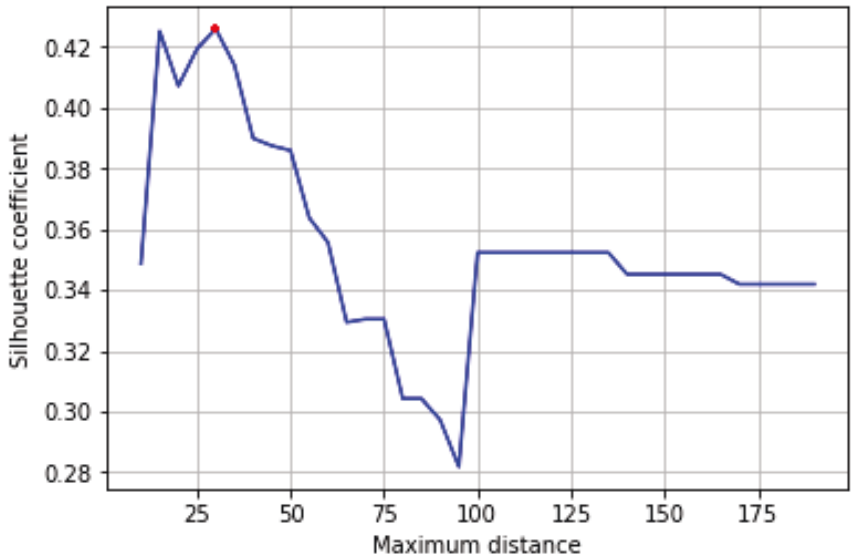

Figure 7.3. The silhouette coefficient for choosing the maximum distance.

The maximum value of the silhouette coefficient was gained at maximum dis-

tance = 30 and is equal to 0.426. As the result of the agglomerative hierarchical clustering with the selected maximum distance, 36 clusters were obtained.

The values of the silhouette coefficient and corresponding numbers of clusters are shown in Fig. 7.4. The level of cutting the dendrogram of hierarchical clustering on the selected distance is presented as a horizontal line in Fig. 7.2.

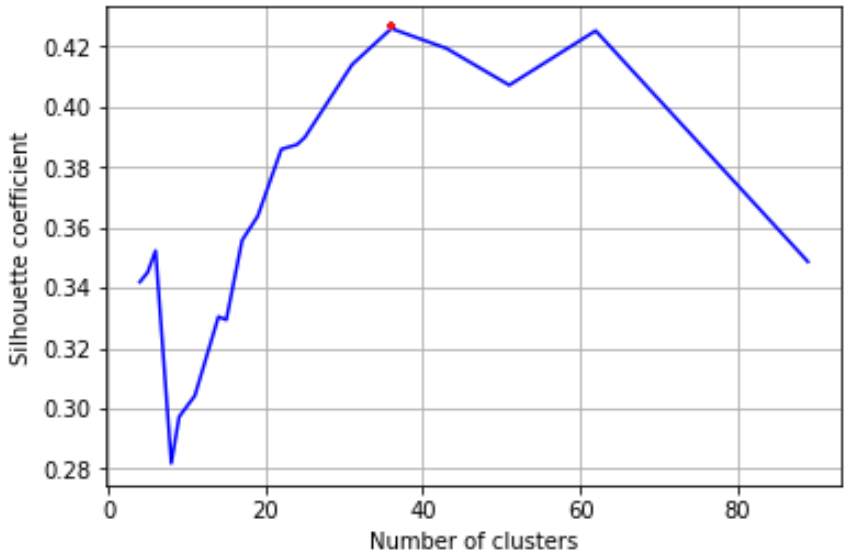

Figure 7.4. The silhouette coefficient and number of clusters.

A human inspection of 36 formed clusters revealed that they allow identification of the patterns of users behaviour by putting similar attempts to produce an attack in the same cluster. The logs from Listing 7.3. represent two similar attempts to get columns from the DB table, after producing a clustering, both of them were put in Cluster 7.

Listing 7.3. Clusters of SQL injections.

```
'" AND 1 = 0 UNION SELECT username, salt, 3, 4, 5 FROM
users;
'" AND 1 = 0 UNION SELECT username, salt, password, 4, 5 FROM
    users;
```

Meanwhile logs presented in Listing 7.4 demonstrate different patterns of behaviour, and they were put in two different clusters (Cluster 27 and Cluster 29).

Listing 7.4. Clusters of SQL injections.

```
' %"; or 1=1';--
'%"; UNION SELECT * FROM users; --
```

By identifying user behaviour patterns, cluster analysis helps to understand how the attackers think, how their attacks evolve and which data they target. The

clustering produced by the system is useful for reducing the workload of a security analyst (production and research honeypots) or a teacher (education honeypots).

7.3 Performance analysis

Along with functional analysis, the evaluation of the performance of the monitoring and analytics system was produced. It was based on requirements presented in Section 3.3.2.

7.3.1 Scalability (2.a.)

The implemented system does not limit the number of honeypots that can be run in the system. Logging of the honeypot system was designed to be automatic, so that after initialisation it does not require an additional configuration to support running of a new honeypot instances.

Each of 28 honeypots that were run during the user study had a preset logging driver that interacted with the monitoring system.

To check that the number of honeypots is not limited by the monitoring system, a testing creation of a large number of honeypots with and without logging was conducted. For that, a bash script was written that runs `docker run` command in a cycle. For both cases, 400 honeypots were started and run simultaneously in the system, as presented in Listing 7.5. The results shows that the monitoring does not influence the amount of running honeypots.

Listing 7.5. Number of honeypots in the system, search is executed by a Docker image

```
$ docker ps | grep mesarpe/exercise_server:v4 | wc -l
400
```

Thus, only the amount of users in the system (for the considered case study) and resources of the server defines how many honeypots can be run. The consumption of resources by the system is discussed in Section 7.4.1.

Therefore, the proposed system supports scalability and can be embedded in honeypot systems that operate with a changing number of honeypot instances.

7.3.2 Reasonable performance (2.b.)

The automated tests were performed to measure whether the implemented monitoring slows down running honeypots.

For that, a simulated user was programmed to perform 15 actions inside a honeypot that required logging. Time that was consumed for performance of these actions was automatically recorded. The simulated user interacted with its own

honeypot, and no other users were interacting with any honeypots during the test.

The tests were divided into two groups, the first one was conducted over a honeypot that contained logging as in the real system. The second one uses a honeypot without logging.

The results are the following: the average time of tests without logging is 0.58 seconds with a standard deviation of 0.02. The average time of tests with logging is 0.56 seconds. The deduction of two average values of performed tests is equals to 0.02 seconds and is within the standard deviation.

Thus, it can be concluded that the implemented monitoring does not slow down the honeypot system.

7.4 Deployability analysis

This section evaluates deployability of the presented solution based on requirements presented in Section 3.3.3.

7.4.1 Low resource cost (3.a.)

The implementation of the monitoring and analytics over the considered honeypot system has not required an addition of supplemental resources.

Disk consumption

Disk consumption of the monitoring system depends on users' activity inside honeypots. However, the usage of customised honeypot logging allows significant reduction of the size of log files. The tested recording of two honeypots during three days showed that the *main.log* file with the malicious activity was 8,3 times smaller in size than the *docker.log* file with all logs (these files were introduced in Section 5.3).

During the user study, the system has created 2.62MB file with malicious logs and 216 KB file with parsed logs. Files with clusters did not exceed 6 KB. Thus, the system does not produce large files during parsing and analysis, and the size of the log file is smaller than in the systems without a customised logging.

Memory consumption

Along with the disk consumption, a memory consumption by the id of the process was measured. The execution of the parsing program on the data produced during the user study has consumed 110.77MB of RAM. The execution of the clustering program depends on the amount of logs of a certain type of attack in the system. The memory consumption of SQL injection clustering (with a predefined maximum

distance) described earlier required to 111.74MB of RAM.

Although the consumption during analysis can notably increase on large amounts of data, the analysis does not need to be continuously executed. It exists to support the work of an analyst, and can be run periodically during the identification of patterns.

7.4.2 Usage of open-source software (3.b.)

During the implementation of the monitoring and analytics system, proprietary software was not utilised. The monitoring system uses Rsyslog and Logrotate system utilities and a logging driver provided by Docker. The analytics system uses only open-source Python libraries, including SciPy and Scikit-learn. Thus, the implementation of the system does not require additional costs for using non-free software.

7.5 Meeting the research goals

This thesis defined and answered the following research questions:

RQ1: *What requirements should a scalable honeypot monitoring and analytics system meet, and how should it be evaluated?*

This research question was answered in Section 3.3, by setting functional, performance and deployability requirements that should be met by a monitoring and analytics system supporting scalability. These requirements were used in Chapter 7 to evaluate the implemented system.

RQ2: *What type of system architecture should the monitoring system use, and how would this be implemented?*

Chapter 5 answered the question, describing a centralised monitoring system that supports scalability and is able to collect data from a large number of honeypot instances.

RQ3: *How the analytics engine can effectively analyse large amounts of data?*

The question was answered by Section 6.1 that describes the analytics system, which supports the work of analyst by automatic deriving of patterns from gathered logs.

RQ4: *What are the possible real-world applications of the monitoring and analytics*

system?

As demonstrated by the case study, one clear use case for a scalable honeypot monitoring and analytics system is to support a large number of educational honeypots in a university course on Information Security. Other uses could include the implementation of the monitoring and analytics in production and research types of honeypot systems, considering variations and extensions discussed in Chapter 8.

8. Discussion

This chapter answers RQ4, discussing possible applications of the system.

8.1 Variations

The proposed solution was implemented over the educational honeypot system. It considers a case with known vulnerabilities, defined types of attacks and with a requirement for scalability. The implementation of the solution on other types of honeypot systems may require additional changes.

If a honeypot system is purposed to find new types of vulnerabilities, it should consider collection of all logs instead of customised logging as was mentioned in Section 5.3. In addition, during the clustering, the filtration of noisy clusters consisting of logs without attacks should be considered.

In cases when it is not possible for a system to differentiate logs by the type of attack, the division of clustering on two steps can be considered: first step, when clusters of attacks are received, and second step, when clusters of patterns are derived from each attack cluster.

8.2 Extensions

8.2.1 Providing feedback

The provided system has a potential for providing a feedback for users in educational honeypot systems. It aims to allow students to receive additional information after an attempt to exploit a vulnerability, and thus assists in studying Information Security.

When an attempt causes an error on a server, the feedback easily can be provided by forwarding this error to a student; in the considered case, to display it on an

attacked web page or reply from an IoT device.

However, in the other case, the task of providing feedback becomes more complicated. One possible direction to consider, taking into account the current functionality of the monitoring and analytics system, is the following. An analyst manually labels clusters, received after the analysis, with potential feedback. Then for each log that has not caused an error, classification is applied.

For that, the distances between the received log and other logs of the same type of attack are measured to get the sample attributes. Then the classification algorithm (for instance, k-nearest neighbours) is trained with clusters received after the initial clustering.

Finally, the appropriate cluster of the received log is predicted by the classification. If the probability of being related to the cluster is high enough, the corresponding label with the feedback can be transmitted to the student.

In this approach, the clustering should be performed periodically when a sufficient amount of new logs has been collected and reviewed by an analyst. The classification runs automatically for every new log of the corresponding type in the system.

8.2.2 Detection of new patterns

Another possible extension is the deriving of new patterns of attacks in the system. If the system does not require a real-time response (probably, research honeypot systems), that can be done during the clusterisation of data. If a sample log has a significant distance from other logs of the same type of the attack or was clustered as a separate cluster without other representatives, it may be concluded that a new pattern was discovered. New patterns can be visualised on a separate page of the dashboard.

For systems that require alerts when new patterns are discovered (probably, the production honeypots), the possible solution is close to one described in the feedback section, which combines clusterisation and classification. If during the real-time classification the log can not be assigned to any cluster with a high enough probability, the alert about discovery of the new pattern can be sent to an analyst.

8.3 User study with analysts

Although the system was evaluated on users, the study with analysts was not conducted. In the proposed solution, an analyst interacts with the dashboard,

reviewing gained statistics and patterns of users behaviour. In addition, an analyst is able to revise clustering by setting the value of the clustering parameter.

Performing the user study with analysts can evaluate the usability of the dashboard and enable collection of data about what information should be shown. In addition, the evaluation of the received patterns of users behaviour by an analyst can assist in making decisions about implementation of the clustering process and the most convenient way to provide the ability to change the clustering parameter value.

9. Related work

Honeypots are widely used [4,5,21,27,29,41,43] for studying adversary behaviour. Thakar et al. [38] presented a monitoring and analytics solution for honeypots. It consists of Data Capture, Data Analysis and Signature Extraction components. The solution covers only monitoring of network traffic of a honeypot and does not consider scalability.

The Data Analysis component provides a graphical interface for an analyst that presents gathered information with statistics by port and IP. As customised logging is not implemented in the system, the interface demonstrates all collected logs that should be filtered manually on this step by the analyst.

The Signature Extraction component analyses filtered data to extract attack signatures. This is done by applying the Longest Common Substring (LCS) algorithm. Although it can result in high-quality signatures, LCS also identifies a large number of impractical signatures of small strings or strings presenting normal operations.

Cabaj et al. [10] proposed two solutions for monitoring and analytics, implemented on different honeypot systems. The first solution applies data mining techniques on gathered from a honeypot data for the automatic discovery of patterns. It does not consider monitoring of honeypots and is focused on an analysis. It only handles connections that reached a honeypot; they consist of such items as source and destination IP address and port, and used protocol. The analysis uses two types of patterns: frequent sets and jumping emerging patterns, their application allows reduction of amount of data that should be revised by an analyst.

A similar solution is presented by Pouget et al. [32] that identifies root causes that can be associated to one attack tool by analysing ports sequence collected from a honeypot. It applies association rule (AR) mining, specifically, the Apriori algorithm.

In contrast, the second solution of Cabaj et al. implements monitoring of web-

application honeypots. It proposed to collect details about data exchange in application layer. It assumes that a PHP script for data capture is put into each web page of a honeypot, and does not consider scalability and implementation of the analysis.

Pawar et al. [30] presented a low-interaction honeypot that captures attacks on the web applications layer, emulating the HTTP and Telnet service.

Although the solution does not provide an analysis, and the log traffic directly goes to the production network, it implements the Graphical User Interface (GUI) for an analyst. It displays gathered logs and related statistics, and is able to provide additional information for the selected IP.

Roesch [33] has suggested the usage of Snort for a honeypot monitoring. It was implemented in a range of honeypot systems, including a low-interaction Java based honeypot presented by Maheswari et al. [24]. Snort is a packet sniffer and logger that performs content pattern matching and can detect a wide range of attacks. It consists of a logging and alerting component, packet decoder and detection engine. However, this approach considers a honeypot only as an intrusion detection mechanism.

Veerasamy et al. [39] presented an analytics system for honeypots. This solution performs a data mining process on traffic, captured from honeypots. First, the data is clustered by some attribute, for instance destination address, and then labelled with meta-data. These clusters are divided into the training and the validation sets. Further, the training set is used to generate the rules by applying the rule induction algorithm. Thus, using a decision tree on the validation set, the system is able to identify the attacks.

The need of data for training is the main limitation of this solution. In addition, the system is aimed at detecting attacks, so it focuses on one type of malicious activity and provides only binary decisions about data for an analyst. In contrast, this thesis solution provides clusters of patterns for all gathered malicious activity.

In summary, previous work does not provide a full solution for the monitoring and analysis of a honeypot system that would meet issues presented in the problem overview section.

10. Conclusion

The proposed system was designed to address the challenge of monitoring honeypot systems with a large number of honeypot instances, meanwhile supporting the work of an analyst with a significant amount of data. The solution consists of three components: monitoring, which is responsible for the collection of logs from all honeypot instances; analysis, which identifies patterns of attacks; and visualisation, which displays the obtained data in a convenient way. Other existing solutions propose systems for monitoring one or a small number of honeypots without considering scalability, and generally analyse one type of attack.

After conducting a user study, it was observed that the monitoring system automatically covers all honeypots in the system, collecting all required logs from them, without reducing the number of honeypots that can be run in the system and without a slowdown. Thus, it provides completeness and scalability of the system. The analytics component clusters gathered data in both labelled and unlabelled ways, identifying patterns of users behaviour with a minimal or no participation of an analyst. Therefore, it allows reducing of the analyst load and provides the ability to explore the actions of attackers, the evolution of their attacks, the common techniques to exploit different types of vulnerabilities and the data targeted by attackers. All data in the system has a structural representation and can be visualised on the dashboard. The system does not have a high resource cost and is implemented using open-source software.

While the solution was implemented on the existing honeypot system and considers such system features as a known range of vulnerabilities and an ability to automatically differentiate types of attacks and users in the system, it can be reconfigured to be applied over systems without these features as discussed in Section 8.1.

Possible extensions of the system were presented in Section 8.2. One of them is adding of the functionality for detection of new patterns of attacks, both in the general case and when real-time alerts are required. Considering the case study

honeypot system, the future work can be focused on the feedback extension. It should assist students of the Information Security course in gaining knowledge about exploiting vulnerabilities provided by honeypots. This extended solution can be applied to other educational systems, including ones presented by Irvine et al. [18] and Cliffe Schreuders et al. [34].

Bibliography

[1] Configure logging drivers. https://docs.docker.com/config/containers/logging/configure/. [Online; accessed 02-April-2018].

[2] The rocket-fast system for log processing. https://www.rsyslog.com/. [Online; accessed 01-April-2018].

[3] Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining text data*, pages 77–128. Springer, 2012.

[4] Eric Alata, Vincent Nicomette, Mohamed Kaâniche, Marc Dacier, and Matthieu Herrb. Lessons learned from the deployment of a high-interaction honeypot. In *Dependable Computing Conference, 2006. EDCC'06. Sixth European*, pages 39–46. IEEE, 2006.

[5] Yaser Alosefer and Omer Rana. Honeyware: a web-based low interaction client honeypot. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 410–417. IEEE, 2010.

[6] S Aranganayagi and K Thangavel. Clustering categorical data using silhouette coefficient as a relocating measure. In *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on*, volume 2, pages 13–17. IEEE, 2007.

[7] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[8] LV Bijuraj. Clustering and its applications. In *Proceedings of National Conference on New Horizons in IT-NCNHIT*, page 169, 2013.

[9] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[10] Krzysztof Cabaj, Marek Denis, and Michał Buda. Management and analytical software for data gathered from HoneyPot system. *Information Systems in Management*, 2, 2013.

[11] Jeeva S Chelladhurai, Vinod Singh, and Pethuru Raj. *Learning Docker*. Packt Publishing Ltd, 2017.

[12] CS-C3130 Information Security Course. Exercise task descriptions. Aalto University, 2017.

[13] Rasmus Dahlberg and Tobias Pulls. Standardized syslog processing: Revisiting secure reliable data transfer and message compression. Karlstads universitet, 2016.

[14] Sergiu Eftimie and Ciprian Racuciu. Honeypot system based on software containers. *Scientific Bulletin" Mircea cel Batran" Naval Academy*, 19(2):582, 2016.

[15] Wenjun Fan, Zhihui Du, David Fernández, and Víctor A Villagrá. Enabling an anatomic view to investigate honeypot systems: A survey. *IEEE Systems Journal*, 2017.

[16] Rainer Gerhards. The syslog protocol. 2009.

[17] Ali Ikinci, Thorsten Holz, Felix C Freiling, et al. Monkey-Spider: Detecting malicious websites with low-interaction honeyclients. In *Sicherheit*, volume 8, pages 407–421, 2008.

[18] Cynthia E Irvine, Michael F Thompson, Michael McCarrin, and Jean Khosalim. Labtainers: a Docker-based framework for cybersecurity labs. 2017.

[19] Nilesh Kakade, Mayur Shinde, Akshay Gawali, and Ajinkya Bhoite. Java based honeypot: Intrusion detection system. 2018.

[20] Karen Kent and Murugiah Souppaya. Guide to computer security log management. *NIST special publication*, 92, 2006.

[21] Iyad Kuwatly, Malek Sraj, Zaid Al Masri, and Hassan Artail. A dynamic honeypot design for intrusion detection. In *Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on*, pages 95–104. IEEE, 2004.

[22] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[23] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013.

[24] V Maheswari and ZPE Sankaranarayanan. Capturing and analysing the intruder attacks using a platform independent honeypot deployment. *Asian J. Inform. Tech*, 6(7):801–805, 2007.

[25] K Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12, 2011.

[26] Iyatiti Mokube and Michele Adams. Honeypots: concepts, approaches, and challenges. In *Proceedings of the 45th annual southeast regional conference*, pages 321–326. ACM, 2007.

[27] Jose Nazario. PhoneyC: A virtual client honeypot. *LEET*, 9:911–919, 2009.

[28] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

[29] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. IoTPOT: analysing the rise of IoT compromises. *EMU*, 9:1, 2015.

[30] Ashwini Pawar, Kimaya Siddhabhati, Sadhana Bhise, and Snehal Tamhane. Web application honeypot. *International Journal*, 2(3), 2014.

[31] Fabien Pouget, Marc Dacier, and Hervé Debar. White paper: honeypot, honeynet, honeytoken: terminological issues. *Rapport technique EURECOM*, 1275, 2003.

[32] Fabien Pouget, Marc Dacier, et al. Honeypot-based forensics. In *AusCERT Asia Pacific Information Technology Security Conference*, 2004.

[33] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[34] Z. Cliffe Schreuders, Thomas Shaw, Mohammad Shan-A-Khuda, Gajendra Ravichandran, Jason Keighley, and Mihai Ordean. Security scenario generator (SecGen): A framework for generating randomly vulnerable rich-scenario VMs for learning computer security and hosting CTF events. In *2017 USENIX Workshop on Advances in Security Education (ASE 17)*, 2017.

[35] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[36] Clifford Stoll. Stalking the wily hacker. *Communications of the ACM*, 31(5):484–497, 1988.

[37] Pang-Ning Tan et al. *Introduction to data mining*. Pearson Education India, 2006.

[38] Urjita Thakar, Sudarshan Varma, and AK Ramani. HoneyAnalyzer–analysis and extraction of intrusion detection patterns & signatures using honeypot. In *Proceedings of the Second International Conference on Innovations in Information Technology*, pages 1–7, 2005.

[39] Namosha Veerasamy, PM Mokhonoana, and Johannes Vorster. Applying data-mining techniques in honeypot analysis. 2006.

[40] Jordan Wright. A simple Elasticsearch honeypot. `https://github.com/jordan-wright/elastichoney`. [Online; accessed 02-April-2018].

[41] Takeshi Yagi, Naoto Tanimoto, Takeo Hariu, and Mitsutaka Itoh. Enhanced attack collection scheme on high-interaction web honeypots. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 81–86. IEEE, 2010.

[42] Li Yujian and Liu Bo. A normalized Levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.

[43] Feng Zhang, Shijie Zhou, Zhiguang Qin, and Jinde Liu. Honeypot: a supplemented active defense system for network security. In *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*, pages 231–235. IEEE, 2003.

[44] Jianwei Zhuge, Thorsten Holz, Xinhui Han, Chengyu Song, and Wei Zou. Collecting autonomous spreading malware using high-interaction honeypots. In *International Conference on Information and Communications Security*, pages 438–451. Springer, 2007.