MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Application of the INJECT platform in hands-on cybersecurity exercises

Bachelor's Thesis

PATRIK MIKUŠ

Brno, Spring 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Application of the INJECT platform in hands-on cybersecurity exercises

Bachelor's Thesis

PATRIK MIKUŠ

Advisor: doc. RNDr. Jan Vykopal, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2025

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Patrik Mikuš

Advisor: doc. RNDr. Jan Vykopal, Ph.D.

Acknowledgements

I would like to thank my supervisor, doc. RNDr. Jan Vykopal, Ph.D., for his invaluable guidance throughout my work. Furthermore, my sincere thanks go to my family, my partner Sára, and her family for their continual support during my studies.

In the preparation of this work, I used ChatGPT for suggestions on vocabulary and text structure. After using this tool, I reviewed the content and take full responsibility for it.

Funded by the European Union under Grant Agreement number 101087529. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Abstract

This thesis investigates the enhancement of practical cybersecurity education by adapting the PB177 Cyber Attacks course exercises for the INJECT Exercise Platform (IXP) and developing a prototype for monitoring students' activity during these exercises. The adaptation process involved transforming the first PB177 lab exercise into IXP's exercise definition format, highlighting missing platform features. These identified features were reported, leading to partial enhancements to IXP. Subsequently, a monitoring prototype was developed to capture, process, and visualize executed commands in the PB177 lab exercises, thereby providing instructors with better insight into students' activity.

Keywords

cybersecurity, INJECT exercise platform, PB177, lab exercises, OpenSearch

Contents

1 Introduction **1**

2 Background **3**
2.1 PB177 Cyber Attacks 3
2.2 The INJECT Exercise Platform 7
2.3 OpenSearch Stack 15

3 Transferring the PB177 exercise to IXP **19**
3.1 Adaptation of the first lab exercise 19
3.2 Identification of missing features 27

4 Prototype development **33**
4.1 Requirements 33
4.2 Software used for the implementation 34
4.3 Implementation 35
4.4 Visualization 49
4.5 Limitations 51

5 Conclusion **55**

A Content of the attachments **56**

Bibliography **57**

List of Figures

2.1 Screenshot of learning objectives from the lab materials . 4
2.2 Screenshot of theoretical concepts from the lab materials . 4
2.3 Screenshot of sandbox instructions from the lab materials 5
2.4 Screenshot of task from the lab materials 5
2.5 Typical structure of the PB177 lab sandboxes 6
2.6 IXP home page 8
2.7 Exercise panel 9
2.8 Trainee view 11
2.9 Instructor view 12
2.10 Analyst view 13
2.11 Data flow in the OpenSearch Stack 15
2.12 Structure of an OpenSearch cluster 18

3.1 Overview of adaptation process 20
3.2 Attempt to include a note in a question 28
3.3 Example of a free-form question in IXP 30
3.4 Example of the *note* field in the IXP 32

4.1 Example of an event in OpenSearch Dashboards 51

1 Introduction

In today's world, digital technologies have become an essential part of everyday life. Although they offer numerous benefits, they also pose significant security risks, including malicious software, cyber fraud, data breaches, and sophisticated cyberattacks. As a result, governments and organizations face these threats daily, which increases the demand for experienced cybersecurity professionals [1].

Many educational institutions focus heavily on theoretical concepts. While understanding the foundational principles of cybersecurity is essential, this approach often creates a gap between academic knowledge and practical application. As a result, students usually complete their studies with a strong theoretical foundation in cybersecurity but struggle to apply their knowledge in real-world scenarios, lacking the hands-on experience necessary to tackle these challenges [2]. One way to bridge this gap is by including more hands-on activities and practical exercises in cybersecurity education.

Masaryk University, through its Faculty of Informatics, offers several cybersecurity courses that emphasize hands-on learning, one of which is the PB177 Cyber Attacks course. This course helps students understand the lifecycle of cyberattacks and standard attack techniques. Additionally, it provides practical experience in executing attacks, analyzing them, and defending against them in a controlled and secure environment [3].

However, a key challenge of this course is the lack of monitoring tools to track students' activity during seminars – specifically, the commands executed by each student during the exercises. Such tools are essential for instructors of the PB177 course to assess each student's performance, identify who is stuck on a particular task, and provide timely assistance. Without these tools, instructors must walk around the classroom and ask each student individually about their progress.

To address this issue, this thesis explores the potential for conducting PB177 exercises within the INJECT Exercise Platform (IXP) – a web-based learning environment designed for delivering, executing, and evaluating tabletop exercises [4]. It identifies missing features necessary for the successful execution of exercises and develops a

prototype of a technical solution that allows instructors to monitor students' activity in real time during seminars.

This thesis is divided into five chapters. Chapter 2 provides an overview of the PB177 Cyber Attacks course, followed by a description of the IXP and the role of OpenSearch Stack [5] in monitoring and analyzing data. Chapter 3 focuses on the process of transferring the PB177 lab exercise to IXP. It covers the adaptation of an existing exercise into IXP, identifies missing features that limit the adaptation and effective execution of the exercise, and discusses the reporting of these missing features. Chapter 4 presents the development of a prototype for monitoring students' activity during the PB177 lab exercises. It describes the defined requirements of the prototype and the software components used for its implementation. It also explains the implementation process, the methods used to visualize and analyze students' activity during the exercises, and the limitations identified during development. Finally, Chapter 5 summarizes the thesis and its results.

2 Background

2.1 PB177 Cyber Attacks

PB177 Cyber Attacks is a hands-on course offered by the Faculty of Informatics [3]. The course consists of in-person lectures and lab sessions, where students practice cyber attack techniques and defense strategies in a controlled environment (sandbox). The course is mandatory for students in the bachelor's *Cybersecurity* program but is also available to other students. To pass the course, students must complete four practical homework assignments focused on applying the acquired knowledge during the semester and one final practical exam at the end of the semester.

Course objectives and topics covered

The main objective of PB177 is to provide students with knowledge and experience related to the lifecycle of cyber attacks. Through hands-on exercises, students explore how cyber attacks are carried out, how to identify them, and how to protect against them. By the end of the course, students are expected to:

- Understand the complete lifecycle of a cyber attack.
- Become familiar with major cybersecurity frameworks and taxonomies, such as MITRE ATT&CK®, MITRE D3FEND™, MITRE ENGAGE™, and the Cyber Kill Chain.
- Execute various attack techniques, analyze them, and defend against them.
- Apply basic concepts of penetration testing and vulnerability management [3].

Existing lab exercises

An important part of PB177 consists of lab exercises, which students complete during seminars. These exercises are text-based, written

in Markdown [6], and stored in a Gitlab repository. They typically consist of the following components:

- Learning objectives, which outline what students should learn from the exercise, see Figure 2.1.

Learning objectives

At the end of this lab session, you will be able to:

- create a wordlist and run attack on SSH password-based authentication,
- discover active accounts and make a lateral movement,
- install Metasploit Framework at a future command and control (C2),
- setup a C2 channel using Metasploit Framework,
- use C2 for collection and exfiltration of target data,
- capture and analyze C2 traffic using tcpdump and Wireshark,
- run a trivial DNS reflection attack and analyze generated packets.

Figure 2.1: Screenshot of learning objectives from the lab materials

- Relevant theoretical concepts, providing the necessary background for the exercise, see Figure 2.2.

▾ **What is privilege escalation?**

Techniques that adversaries use to gain higher-level permissions on a system or network.
See MITRE ATT&CK® Matrix for Enterprise for more info.

▾ **What is defense evasion?**

Techniques that adversaries use to avoid detection throughout their compromise.
See MITRE ATT&CK® Matrix for Enterprise for more info.

▾ **What is credential access?**

Techniques for stealing credentials like account names and passwords.
See MITRE ATT&CK® Matrix for Enterprise for more info.

Figure 2.2: Screenshot of theoretical concepts from the lab materials

- Instructions on how to start the sandbox, see Figure 2.3.

Start the sandbox

1. Update the repo by running `git pull` in `labs` directory. Go to `3` directory.
2. Login to get access to Podman images: `podman login gitlab.fi.muni.cz:5050`.
3. **Run `podman-compose pull` to download images used in this lab.**
4. Run `podman-compose up -d`.
5. Check you can access Kali container using terminal and RDP. This lab uses two containers, `podman-compose ps -q | wc -l` should display `2`.

Figure 2.3: Screenshot of sandbox instructions from the lab materials

- The main topics of the exercise, each containing multiple tasks and a solution for each task, see Figure 2.4.

Nmap

Nmap ("Network Mapper") is a free and open source utility for network discovery and security auditing.

Task: Scanning IP Blocks

T1595.001, also T1590.005.

Two organization you would like to attack have been assigned IP ranges `10.0.33.0/24` and `10.0.66.0/24`.

Find IP addresses of hosts that are up.

▸ **Solution and follow-up tasks**

Figure 2.4: Screenshot of task from the lab materials

Before starting a specific lab exercise, students must deploy the corresponding sandbox on the *Nymfe* machine in the lab or on their own machine. These sandboxes are composed of multiple containers orchestrated by *Podman* [7]. As shown in Figure 2.5, they typically include an *attacker* container, which serves as the environment where

students execute attacks. This container has all the necessary tools pre-installed and ready to use. Additionally, there is a *server* container, which acts as the target and allows students to practice various defensive techniques.

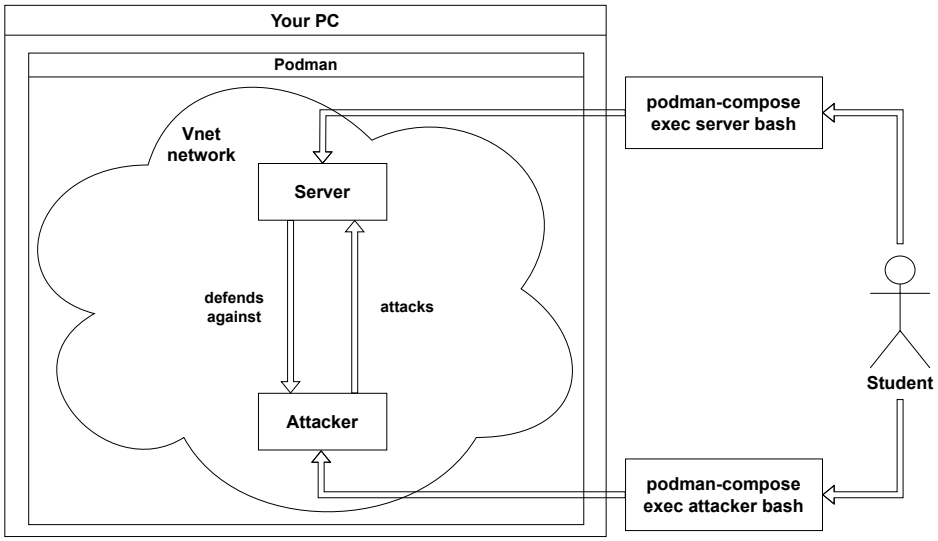

Figure 2.5: Typical structure of the PB177 lab sandboxes

However, the current setup faces several limitations. In particular, there are no built-in monitoring features to track student activity inside sandboxes during seminars. As a result, instructors must walk around the classroom and ask students individually about their progress. This is currently the only way to determine who is making progress, who needs help, or who is stuck on a particular task.

It should also be noted that while Podman is used to orchestrate the containers in the PB177 lab exercises, *Docker* [8] is used in this thesis for implementation purposes. Since Podman is largely compatible with Docker – supporting Docker images and commands – these terms can be used interchangeably in this thesis.

2.2 The INJECT Exercise Platform

The IXP is developed by Masaryk University, and its main objective is to prepare participants for a cybersecurity crises by testing their response actions in various simulated scenarios. The platform provides immediate reactions to participants' decisions and enables detailed monitoring and evaluation of their actions [4].

IXP terminology

This subsection explains the key terminology used in the IXP.

- **Exercise definition** – a ZIP archive from which an exercise in IXP is created. It consists of multiple YAML files [9] that define the main logic and core structure of the exercise.
- **Exercise designer** – the person who creates the exercise scenario and transforms it into an exercise definition.
- **Trainee** – a participant in the exercise for whom the exercise is created.
- **Instructor** – the person who manages the exercise during its execution. This includes monitoring trainees' progress, responding to emails, sending notifications, and helping stuck teams. The instructor also addresses issues not covered by the predefined scenario and provides guidance throughout the entire exercise to ensure its progress.
- **Inject** – predefined messages or emails delivered to trainees during the exercise. They move the exercise forward by providing general information, tasks, emails, or notifications. Injects enable a dynamic and flexible scenario where events unfold based on trainees' actions, and the injects they receive.
- **Milestones** – binary variables that trigger actions within the exercise, such as sending injects, emails, or displaying new questions. A milestone consists of a name and a boolean value indicating whether it has been reached. Milestones are typically activated or deactivated by the trainees' actions, often through tool usage or submitted answers.

- **Tools** – components used for interaction with trainees. They can serve various purposes. Exercise designers can define as many tools as needed and specify the behavior of each tool. The tools are automated and available to trainees throughout the entire exercise.

Overview of IXP components

The IXP provides a comprehensive and user-friendly interface for managing users, instructing or performing tabletop exercises, and analyzing their results. Figure 2.6 presents the home page for instructors and administrators. This page includes several components that enable them to perform various actions, discussed in the following sections.

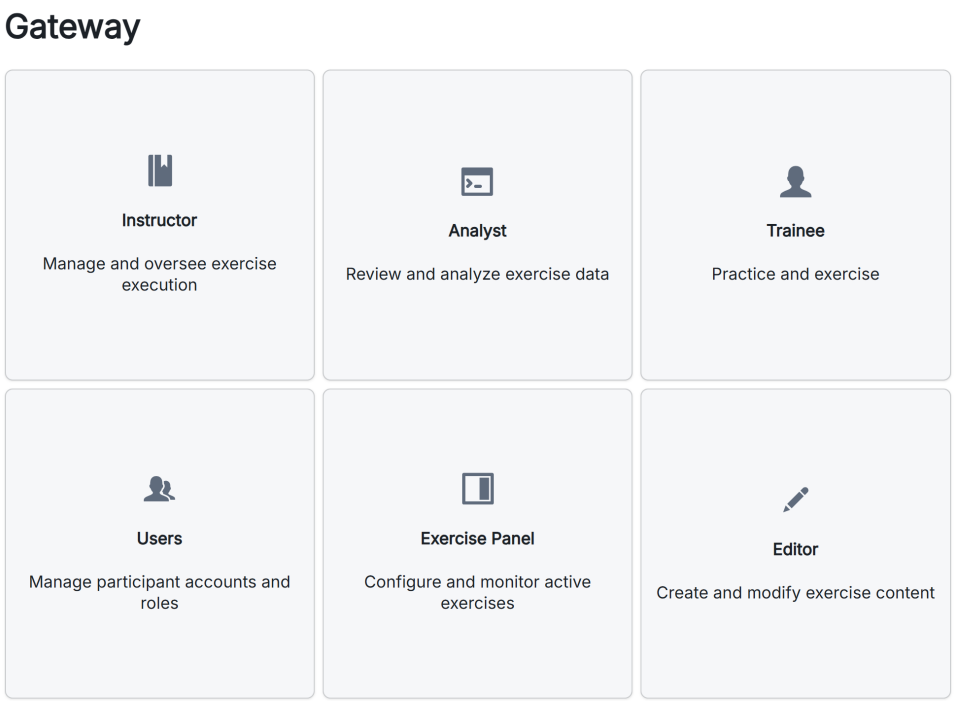

Figure 2.6: IXP home page

Exercise Panel

The *Exercise Panel* serves as an interface for managing exercises. It allows exercise designers to upload exercise definitions and create exercises based on those definitions. The panel supports storing multiple definitions and exercises simultaneously, but only one exercise can be running at a time. Before creating an exercise, the exercise designer can also specify the number of teams the exercise is intended for. Additionally, it also provides the option to download logs from selected exercises. The layout of the exercise panel is shown in Figure 2.7.

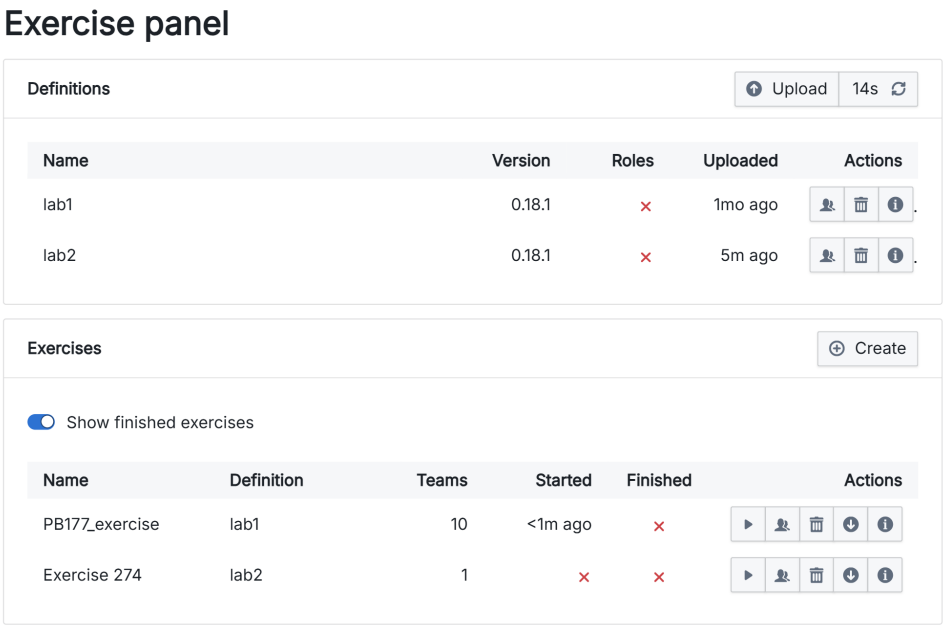

Figure 2.7: Exercise panel

Trainee view

Figure 2.8 depicts the *Trainee* view, which is designed for participants engaging in the exercises. It allows them to access and participate in the exercises, practice response actions, and interact with the environment [10]. This view includes several key tabs:

- The *General* tab provides an overview of the current exercise, including its objectives and instructions. It is the starting point for trainees to understand the exercise's goals and expected outcomes.
- The *Tools* tab contains all the necessary tools required for the exercise. Each tool includes an input field that accepts a predefined set of arguments and a brief description above the input field explaining the tool's purpose. After entering the appropriate arguments, the trainee can submit them using the *Submit* button. The tool's output is then displayed in the middle section of the page.
- The *Emails* tab displays all relevant communications during the exercise. Trainees can interact with simulated characters by exchanging information via emails, which can influence their decisions.
- The *Questions* tab allows instructors to ask trainees various questions during the exercise. These may include theoretical questions or requests for feedback on the exercise.

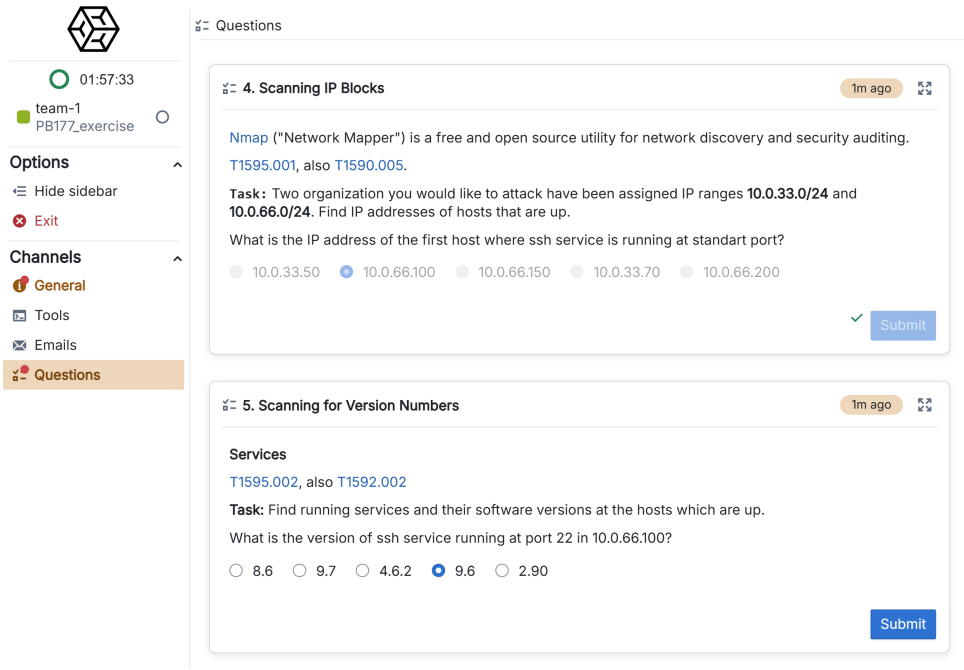

Figure 2.8: Trainee view

Instructor view

Figure 2.9 presents the *Instructor* view. It is designed similarly to the Trainee view but with key differences to support the instructor's role. This view provides an organized interface for managing and overseeing the exercise [10]. The main page contains several key elements:

- *Open/close milestones* tab – Allows instructors to manage milestones by making them as reached or unreached in real time.
- *Send an inject* tab – Allows instructors to send custom injects to trainees during the exercise.
- *Select teams* tab – Allows instructors to choose which teams they want to manage during the exercise. Once selected, they can view specific details for each team, such as which injects a team has received, which milestones it has reached, the current question being answered, and the tools used. Instructors can switch

between teams using the left sidebar under the *Teams* section or through the team tabs at the top of the interface. This setup enables them to efficiently monitor and guide multiple teams simultaneously.

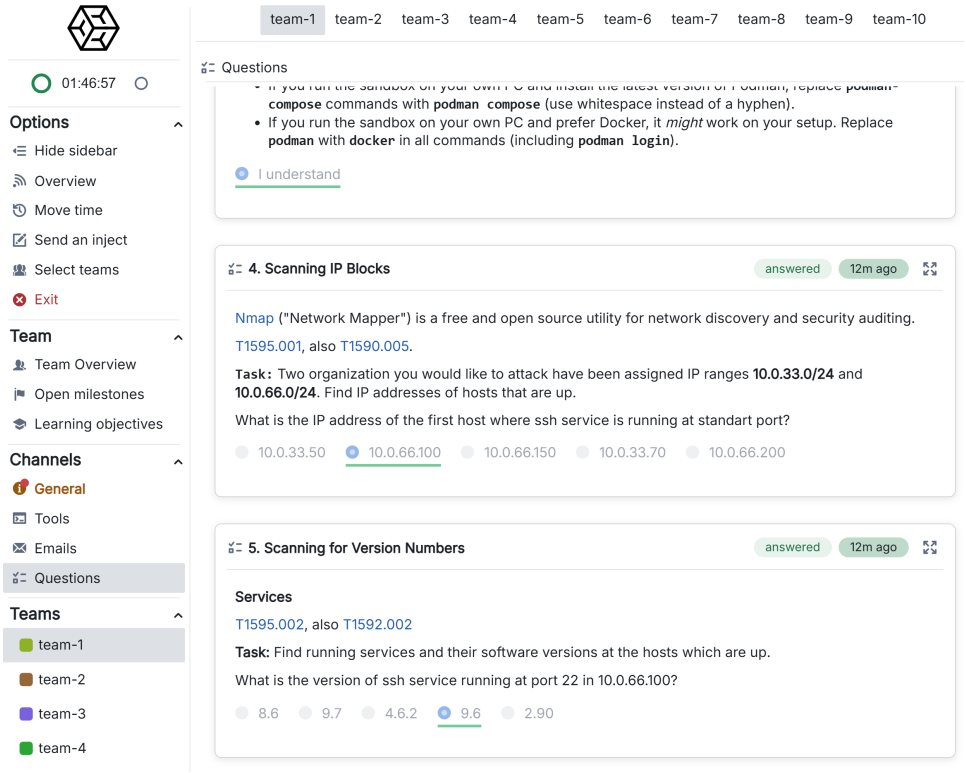

Figure 2.9: Instructor view

Analyst view

Figure 2.10 illustrates the *Analyst* view. This view allows instructors to review and analyze the data generated during exercises. Instructors can access logs, data visualizations, and other metrics to evaluate the trainees' performance and the overall results of the exercise [10]. The Analyst view includes these tabs:

- The *Dashboard* offers visualizations of reached milestones, received injects, received emails, and tools used.
- The *Milestones* tab is used to track which milestones were reached by each team.
- The *Learning objectives* tab allows instructors to see which learning objectives have already been achieved by the trainees.
- The *Emails* tab provides insights into which emails were sent and received by the teams.
- The *Tools* tab enables the analyst to review which tools were used by each team.
- The *Questionnaires* tab contains the responses to all questions posed to the trainees during the exercise.

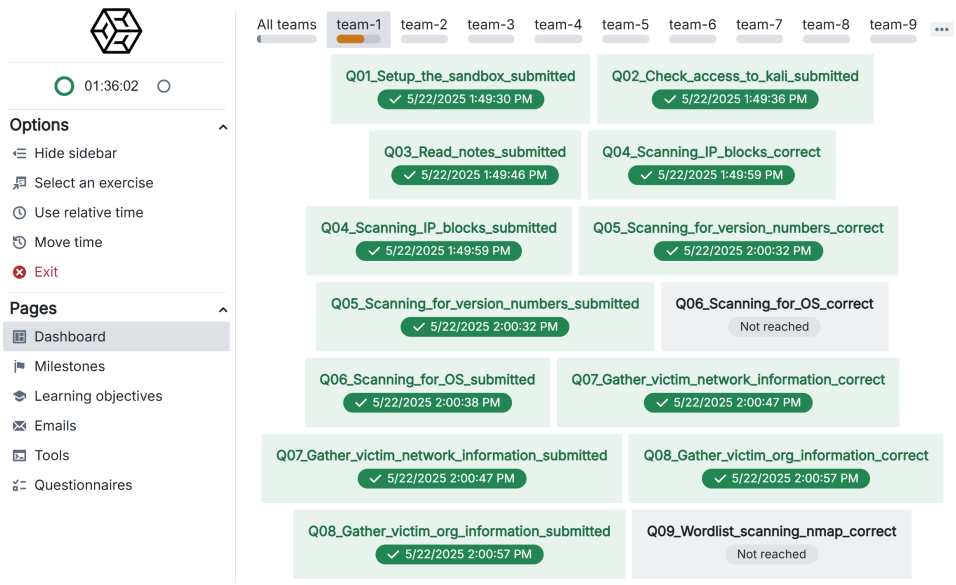

Figure 2.10: Analyst view

Exercise definitions

An exercise in IXP can be created only from an exercise definition, which consists of the following files:

- *config.yml* – Contains general configuration settings for the exercise, such as exercise duration, description, and format version.
- *email.yml* – Defines the configuration for all email addresses used within the exercise. Each email address entry includes several fields that specify how emails function during the exercise.
- *tools.yml* – Defines tools with specific behaviors. These tools operate by comparing an input argument to predefined parameters. If the argument matches, the tool returns a response; otherwise, it returns a default response. The file specifies various attributes for each tool, such as name, description, response, default response and hint.
- *milestones.yml* – Defines all milestones used in the exercise.
- *questionnaires.yml* – Includes the definitions of all questions used in the exercise.
- *injects.yml* – Contains the definitions of all injects used in the exercise. This file allows instructors to define how and when specific information is delivered to trainees.

The ZIP archive can also contain two optional directories:

- *files* – Contains files that are delivered during the exercise to support the exercise scenario.
- *content* – Contains Markdown files that define the content of emails, questions, and injects.

A single exercise definition can be used to create multiple exercise instances.

2.3 OpenSearch Stack

OpenSearch Stack is an open-source set of tools designed for real-time search, analytics, and observability [5]. It enables users to collect, process, store, search, and visualize large volumes of data. It was originally derived from the Elasticsearch and Kibana projects. The OpenSearch Stack includes Beats, Logstash, OpenSearch, and OpenSearch Dashboards, providing a comprehensive set of tools for data collection, processing, storage, analysis, and visualization.

Figure 2.11 illustrates the overall functioning of the OpenSearch Stack. First, Beats collect logs from a web server and forward them to Logstash. Logstash processes the logs, extracts useful information, and sends the structured output to OpenSearch for indexing and storage. Once the data is stored, users can access OpenSearch Dashboards to visualize and analyze the data through graphs and dashboards.

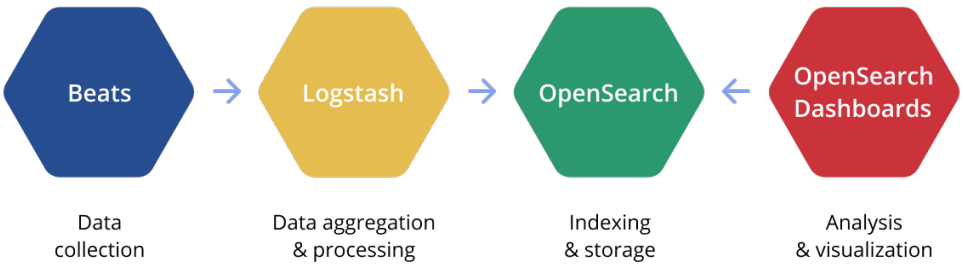

Figure 2.11: Data flow in the OpenSearch Stack

Beats

Beats are lightweight data shippers designed to send various types of data from one or many machines to Logstash, ElasticSearch or OpenSearch [11]. This thesis uses *Filebeat* and *Auditbeat*.

Filebeat is a lightweight shipper for forwarding log data. It monitors log files from specified directories and forwards the logs to defined destinations. It consumes minimal resources and supports a wide range of modules for different log types [12].

Auditbeat is another lightweight data shipper designed to collect and forward activities of users and processes on a system. It integrates directly with the *Linux Audit Framework*, a kernel-level auditing system that monitors and records system events such as file accesses, system calls, and user activities [13]. Through this integration, Auditbeat captures these events in real-time, processes them, and forwards them to OpenSearch, Logstash, or other specified destinations [14].

Logstash

Logstash is a real-time event processing engine [15]. It can receive events from various sources, process them, extract useful information such as timestamps, IP addresses, or HTTP status codes, and then forward the structured output to specified destinations. Although originally developed for processing log data, Logstash has evolved to handle multiple event formats, including JSON and XML.

Its core functionality is based on a pipeline composed of three main stages:

- *Input* – This stage configures how Logstash receives data. It supports numerous input plugins, such as *file*, *syslog*, *beats*, and others. These plugins collect and pass events to the filter.
- *Filter* – In this stage, Logstash processes and transforms the collected data. It offers a rich set of filter plugins to parse, enrich, and modify events. For example, the *grok* filter can parse unstructured log data into structured fields, while the *mutate* filter can add, remove, or modify fields. Filters are applied sequentially.
- *Output* – This stage is responsible for forwarding the processed data to specified destinations. Logstash supports a wide range of output plugins, allowing data to be sent to destinations such as OpenSearch, Elasticsearch, files, stdout, and more.

Logstash's architecture is highly extensible, offering over 200 built-in plugins that allow users to tailor the pipeline to their specific needs. Apart from that, Logstash allows users to use community plugins or develop custom plugins. This flexibility makes it suitable for a wide range of data processing tasks, from simple log collection to complex event processing [15].

OpenSearch

OpenSearch is the central component of the OpenSearch Stack, responsible for storing, indexing, and searching data. It is a distributed, open-source search and analytics engine based on the *Apache Lucene* library. After adding data to OpenSearch, users can search by fields, sort results, or aggregate data for analysis [16].

An OpenSearch node represents a single instance of the OpenSearch engine. Multiple nodes can form an OpenSearch cluster, as shown in Figure 2.12. Clusters allow nodes to scale horizontally to handle increased load and storage demands. OpenSearch can operate either as a single-node cluster – an individual instance of the OpenSearch engine, or as a multi-node cluster, where responsibilities such as indexing, storing, and searching are distributed across multiple nodes [17].

Data in OpenSearch is organized into *indices*, which are collections of documents. Each document represents data stored in JSON format. To distribute data across nodes in a cluster, indices can be divided into shards. Each shard stores a subset of the documents in an index. For example, a 400 GB index might be too large for a single node to handle, but when split into 10 shards of 40 GB each, OpenSearch can distribute them across 10 nodes and manage each shard individually.

Shard can be either a *primary* (original) shard or a *replica* (copy) shard. Replica shards act as backups if a node fails by assigning them to nodes other than their corresponding primary shards. In this way, they provide redundancy but also help to improve search performance, ensuring high availability and scalability.

Additionally, indices use an *inverted index*, which maps words to the documents in which they appear. This feature enables users to search and analyze data [18].

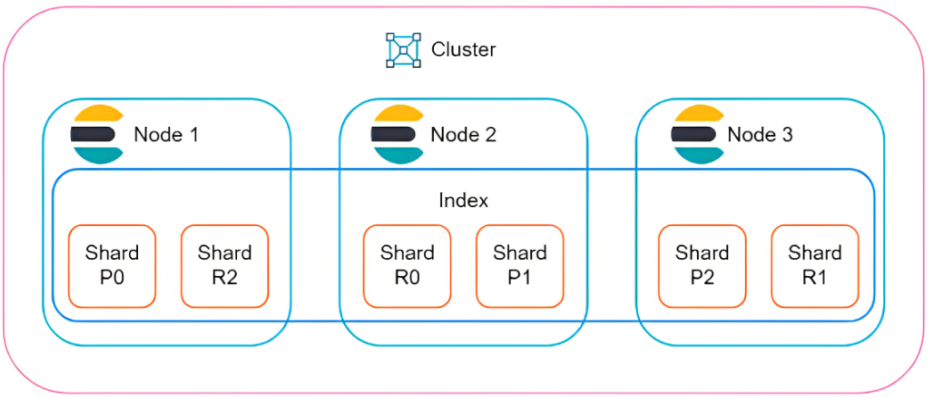

Figure 2.12: Structure of an OpenSearch cluster

OpenSearch Dashboards

OpenSearch Dashboards is a data visualization tool that allows users to explore, analyze, and interact with data stored in OpenSearch. It provides a user-friendly interface for creating dashboards, charts, and graphs, which enables users to analyze and monitor their data effectively. Additionally, OpenSearch Dashboards allows users to manage their OpenSearch clusters by providing insights into cluster health and performance [19].

3 Transferring the PB177 exercise to IXP

For this thesis, we selected one exercise from the PB177 Cyber Attacks course, specifically the first lab exercise. The objective was to transfer this exercise into the IXP and identify the missing features required for its effective execution.

The first lab exercise guides students through a series of offensive cybersecurity tasks, primarily focusing on reconnaissance and scanning techniques. It begins with network scanning using the *Nmap* tool [20], where students explore various methods for detecting open network ports, identifying operating systems, and discovering running services along with their versions. The exercise also focuses on gathering useful information about target networks and performing web content enumeration using tools like *Dirb* [21]. Furthermore, students are encouraged to engage in social engineering by generating phishing emails using AI tools and researching publicly available information to craft spear-phishing campaigns.

3.1 Adaptation of the first lab exercise

To successfully transfer the first lab exercise to the IXP's exercise definition format, it was necessary to adapt the original exercise to the structure required by the platform. An overview of this adaptation process is illustrated in Figure 3.1. It was divided into four main parts, each addressing a different aspect of the exercise configuration.

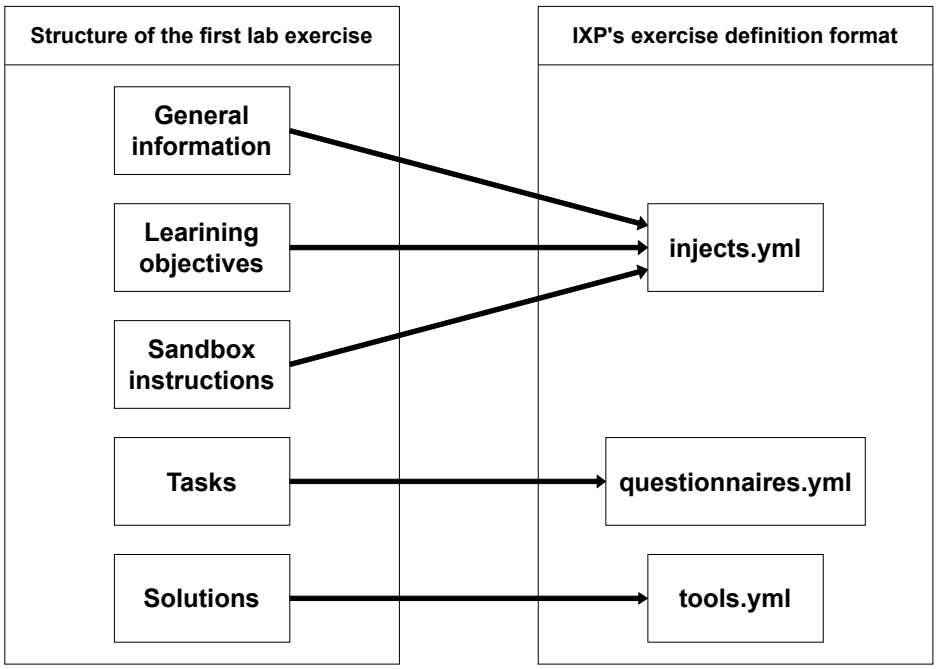

Figure 3.1: Overview of adaptation process

Setting configuration parameters

First, I had to configure the parameters for the IXP exercise in the exercise definition. These parameters are defined in the *config.yml* file, shown in Listing 1.

```yaml
exercise_duration: 120
enable_email: True
email_between_teams: False
team_file_upload: False
team_visible_milestones: False
show_exercise_time: True
enable_roles: False
version: 0.18.1
```

Listing 1: Configuration parameters in *config.yml* file

The exercise duration is set to 2 hours, as the seminar should not exceed this time limit. Almost all other parameters are set to *False*, as trainees do not need them to perform the PB177 lab exercises. There are only two relevant parameters for trainees, which are set to *True*:

- **Enabling in-platform email functionality** allows students to contact the instructor using the platform's email system. This can be useful if students need help and are not present at the seminar or if they wish to clarify something. Instructors can also use it to send relevant information to students.
- **Displaying the remaining duration of the exercise** helps students keep track of the remaining time so that they can complete all tasks within the time limit.

Finally, the exercise definition format specifies its semantic version at the end of the file. When I began the practical part of this thesis, the latest version available was 0.12.1, which I initially used. Later, I upgraded to version 0.18.1, which is the latest version available at the time of writing this thesis.

Transferring the tasks

To transfer the tasks of the first lab exercise into IXP, I decided to map each task into a question in the *questionnaires.yml* file. This approach was the most suitable, as it allows exercise designers to specify answer

options for each question and evaluate submitted responses, helping verify that the task was completed successfully. Listing 2 shows the structure of *questionnaires.yml*, where each task corresponds to one question. The content of each question is stored in the *content* directory. Each question includes several answer options specified in the *labels* field, while the correct answer is defined in the *correct* field.

```
- title: 4. Scanning IP Blocks
  control: { milestone_condition: Q03_Read_notes_submitted,
  activate_milestone: Q04_Scanning_IP_blocks_submitted }
  questions:
    - content:
        content_path: questions/04_scanning_IP_blocks.md
      max: 5
      labels: 10.0.33.50, 10.0.66.100, 10.0.66.150,
      10.0.33.70, 10.0.66.200
      correct: 2
      controls:
        2: { activate_milestone: Q04_Scanning_IP_blocks_
        correct }

- title: 5. Scanning for Version Numbers
  control: { milestone_condition: Q04_Scanning_IP_blocks_
  submitted, activate_milestone: Q05_Scanning_for_version_
  numbers_submitted }
  questions:
    - content:
        content_path: questions/05_scanning_version_
        numbers.md
      max: 5
      labels: 8.6, 9.7, 4.6.2, 9.6, 2.90
      correct: 4
      controls:
        4: { activate_milestone: Q05_Scanning_for_version_
        numbers_correct }
```

Listing 2: Structure of *questionnaires.yml*

A limitation of the original lab exercise is that instructors cannot connect the result of one task to the next because all tasks are presented at once. In contrast, questions in *questionnaires.yml* are displayed sequentially, meaning that the next question appears only after the previous one is submitted. This behavior is achieved using **milestones** within the *control* field.

The *control* field acts as a condition for displaying the question. Within this field, I use two actions:

- *milestone_condition*, which specifies the milestone that must be reached for the question to be displayed. If the milestone is not reached, the question remains hidden.
- *activate_milestone* activates the specified milestone when the question is submitted. This activation indicates that the question has been submitted and allows the following question to be displayed.

Both of these actions use milestones defined in the *milestones.yml* file, specifically those from the first group with the suffix *submitted*, as shown in Listing 3.

The *controls* field at the end of each question enables the exercise designer to define an action for each answer option. When a trainee submits the selected option, the corresponding action specified for that option is executed. However, in all questions, I define actions – specifically *activate_milestone* – only for the correct answers. This means that when a correct answer is submitted, the milestone assigned to the correct answer is activated. In this way, instructors are able to track correct answers throughout the exercise. These milestones are defined in the second group of the *milestones.yml* file, with the suffix *correct*, as presented in Listing 3. If a trainee submits an incorrect answer, the corresponding milestone with the suffix *correct* is not activated, indicating the answer is incorrect. In this case, the trainee cannot answer the question again.

```yaml
# Milestones for submitted answers
- name: Q01_Setup_the_sandbox_submitted
- name: Q02_Check_access_to_kali_submitted
- name: Q03_Read_notes_submitted
- name: Q04_Scanning_IP_blocks_submitted
- name: Q05_Scanning_for_version_numbers_submitted
- name: Q06_Scanning_for_OS_submitted
- name: Q07_Gather_victim_network_information_submitted
- name: Q08_Gather_victim_org_information_submitted
- name: Q09_Wordlist_scanning_nmap_submitted
- name: Q10_Wordlist_scanning_dirb_submitted
- name: Q11_Phishing_for_information_submitted
- name: Q12_Search_open_technical_databases_submitted
- name: Q13_Search_victim_owned_websites_part1_submitted
- name: Q14_Search_victim_owned_websites_part2_submitted
  final: True

# Milestones for correct answers
- name: Q04_Scanning_IP_blocks_correct
- name: Q05_Scanning_for_version_numbers_correct
- name: Q06_Scanning_for_OS_correct
- name: Q07_Gather_victim_network_information_correct
- name: Q08_Gather_victim_org_information_correct
- name: Q09_Wordlist_scanning_nmap_correct
- name: Q10_Wordlist_scanning_dirb_correct
- name: Q12_Search_open_technical_databases_correct
- name: Q13_Search_victim_owned_websites_part1_correct
- name: Q14_Search_victim_owned_websites_part2_correct
```

Listing 3: Structure of *milestones.yml*

Transferring the general information

I decided to transfer the general information – including the learning objectives, theoretical concepts, and sandbox instructions – as an inject at the beginning of the exercise. This inject is illustrated in Listing 4. First, I created a Markdown file named *General_info.md* in the *content*

directory, where I copied the text from the first lab exercise. Next, I created an inject in the *injects.yml* file that delivers the contents of *General_info.md* to the *General* channel at the start of the exercise.

```
- name: General Learning Objectives
  type: info
  target: General
  alternatives:
    - name: General Info
      content: { content_path: injects/General_info.md }
```

Listing 4: Inject for sending general information

The *injects.yml* file also contains injects that act as reminders, presented in Listing 5. These reminders appear in the *General* channel during the exercise when students are stuck on a particular task. This is achieved using the *delay* field, which specifies the time in minutes by which the reminder is delayed before being sent. If students reach a given question and do not submit the answer within the specified time, a reminder is sent, prompting them to use the provided solution and proceed to the next task to complete the exercise on time. The content of each reminder is stored in the *content* directory.

```
- name: Reminder_Q4
  delay: 8
  target: General
  alternatives:
    - name: Reminder
      control: { milestone_condition: Q03_Read_notes_
      submitted and not Q04_Scanning_IP_blocks_submitted }
      content: { content_path: injects/Q04_reminder.md }
```

Listing 5: Injects for sending reminders

Listing 6 depicts the final part of the *injects.yml* file, which contains an inject that informs students upon completing the entire exercise. This inject is triggered by the final milestone, which is activated when students submit the last question.

```
- name: End of exercise
  type: info
  target: General
  alternatives:
    - name: End of exercise message
      control: { milestone_condition: Q14_Search_victim_
      owned_websites_part2_submitted }
      content: { content_path: injects/End_message.md }
```

Listing 6: Inject for informing about exercise completion

Transferring the solutions

There were several possible options for transferring the solutions into the exercise definition. The first option was to place each solution directly below its corresponding question, as in the original PB177 exercise. However, since the exercise definition does not support expandable text, the solution would be displayed together with the question, which is not acceptable.

The second option was to send the solution as a separate question without answer options after each question. However, I found this approach unsatisfactory, as the exercise definition requires at least one answer option for each question, and trainees might not always want to view the solution to every question.

Therefore, I decided to use the tools feature available in IXP. Listing 7 shows the contents of the *tools.yml* file, which defines a single tool for the entire exercise. This tool allows trainees to show the solution to any question on demand. The content of the solutions is stored in the *content* directory. The tool is configured so that when a trainee enters a question number, it checks whether the entered number matches

any of the numbers defined in the tool. If it does, the corresponding solution, specified in the *content* field, is displayed. If the entered number does not match any of the numbers, a default response is shown, informing the trainee that the entered number is incorrect.

```yaml
- name: Show solution
  tooltip_description: Enter the question number to show
  solution of that question.
  hint: Question number
  default_response: { content: Incorrect number }
  responses:
    - param: '1'
      content: { content_path: solutions/No_solution.md }
    - param: '2'
      content: { content_path: solutions/No_solution.md }
    - param: '3'
      content: { content_path: solutions/No_solution.md }
    - param: '4'
      content: { content_path: solutions/Q04_solution.md }
    - param: '5'
      content: { content_path: solutions/Q05_solution.md }
```

Listing 7: Structure of *tools.yml* – solutions are provided for questions 4 and 5, while questions 1-3 display a message indicating no solution is required.

3.2 Identification of missing features

The first identified missing feature is discussed in Section 3.1, where I used the questionnaire feature in IXP to deliver each task from the first lab exercise as a question. The questionnaire allows trainees only to select the options they believe are correct and submit their answers. However, this approach is not sufficient, as it enables trainees to guess the correct answer, thereby reducing the educational value of the exercise. A more effective solution would allow trainees to enter free-text

answers for each question, which could then be evaluated accordingly. This approach would reduce guessing and promote deeper engagement in the exercise. However, this functionality was not available in IXP, whose backend supported exercise definition version 0.12.1 at the time.

The second missing feature was also found in the *quesionnaires.yml* file. It did not allow for the inclusion of a note below the answer options unless it was added as a separate question with at least one answer option, as shown in Figure 3.2.

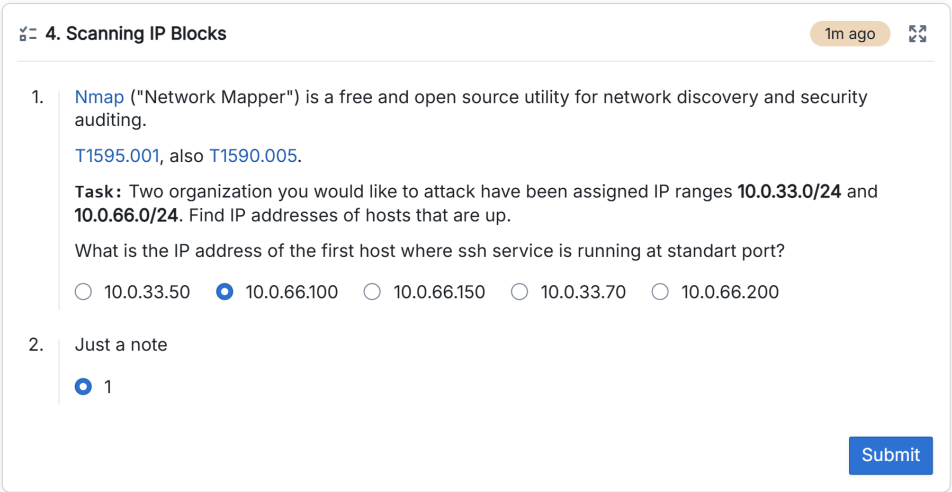

Figure 3.2: Attempt to include a note in a question

Such notes could be useful for providing hints, explanations, or follow-up instructions related to the question. However, the solution described above was not sufficient, as trainees were required to select an answer option for the note as well, which could lead to confusion and reduce the clarity of the exercise. The corresponding YAML code is presented in Listing 8.

```yaml
- title: 4. Scanning IP Blocks
  control: { milestone_condition: Q03_Read_notes_submitted,
  activate_milestone: Q04_Scanning_IP_blocks_submitted }
  questions:
    - content:
        content_path: questions/04_scanning_IP_blocks.md
      max: 5
      labels: 10.0.33.50, 10.0.66.100, 10.0.66.150,
      10.0.33.70, 10.0.66.200
      correct: 2
      controls:
        2: { activate_milestone: Q04_Scanning_IP_blocks_
        correct }
    - content:
        content_path: questions/note.md
      max: 1
```

Listing 8: YAML code of attempt to include a note

The purpose of this code was to try adding additional text below the answer options. However, the *max* field – which defines the number of answer options, was required for both the main question and the note. If it was omitted from the note, the question became invalid.

Although the note could be placed above the answer options, this was not optimal. Since the note serves as additional information, it makes the most sense to place it below the answer options, clearly separating it from the question's main content, which was impossible.

Reporting missing features

To address these missing features, I reported two issues to the IXP development team.

The first issue focused on enabling free-text answers to questions. As a result, the question structure was extended in exercise definition version 0.15.0 to support two types:

- *radio* – the default question type, allowing trainees to select from predefined answer options.
- *free-form* – a newly introduced type that allows trainees to enter arbitrary text as their answer, as illustrated in Figure 3.3. The corresponding YAML structure is shown in Listing 9. A *free-form* question includes the *related_milestones* field, which lists associated milestones to be displayed to the instructor for evaluation, and the *multiline* field, a boolean variable indicating whether the input should support multiple lines of text.

However, automatic evaluation of *free-form* answers is not yet supported in the latest version of the exercise definition (0.18.1), and it remains a future enhancement.

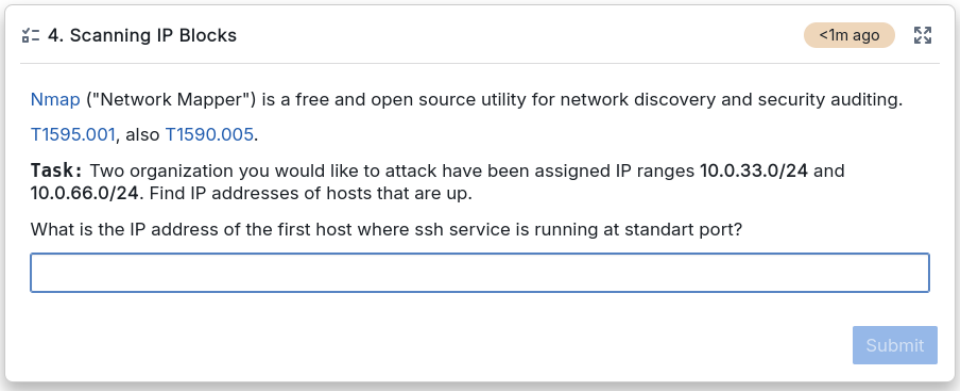

Figure 3.3: Example of a free-form question in IXP

```yaml
- title: 4. Scanning IP Blocks
  control: { milestone_condition: Q03_Read_notes_submitted,
  activate_milestone: Q04_Scanning_IP_blocks_submitted }
  questions:
    - content:
        content_path: questions/04_scanning_IP_blocks.md
      type: free-form
      multiline: False
      related_milestones: [ Q04_Scanning_IP_blocks_
      correct ]
```

Listing 9: YAML code of a free-form question

The second report addressed the limitation of being unable to place notes below answer options. As a solution, an optional *note* field was introduced in IXP's exercise definition version 0.18.1 for use within questions. The structure of this enhancement is shown in Listing 10. As previously discussed, the *note* field allows exercise designers to append additional text to a question. Figure 3.4 presents how this field will look in the IXP.

```
- title: 4. Scanning IP Blocks
  control: { milestone_condition: Q03_Read_notes_submitted,
  activate_milestone: Q04_Scanning_IP_blocks_submitted }
  questions:
    - content:
        content_path: questions/04_scanning_IP_blocks.md
      max: 5
      labels: 10.0.33.50, 10.0.66.100, 10.0.66.150,
      10.0.33.70, 10.0.66.200
      correct: 2
      controls:
        2: { activate_milestone: Q04_Scanning_IP_blocks_
        correct }
      note: When trying to find the right combination of
      switches and tweaking parameters, it is more time
      efficient to target a specific host instead of whole
      network(s).
```

Listing 10: Structure of a question using the *note* field

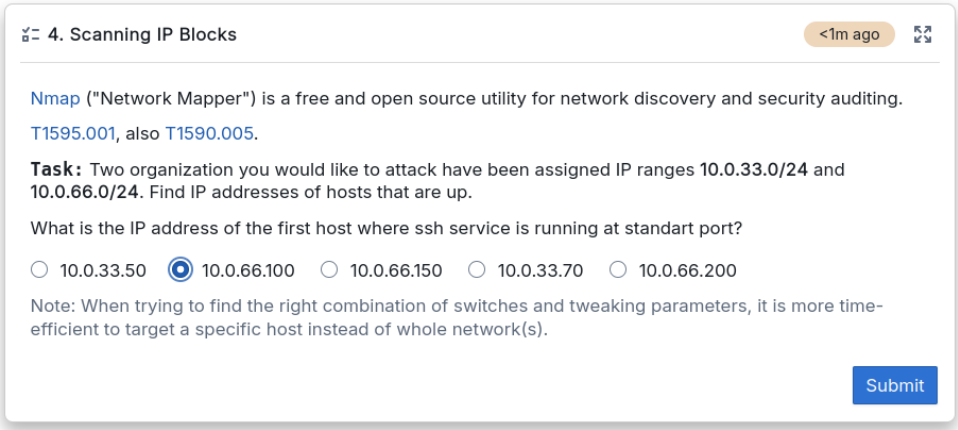

Figure 3.4: Example of the *note* field in the IXP

4 Prototype development

This chapter describes the development of a prototype that enables instructors of the PB177 course to analyze the activity of students in their sandboxes during lab exercises. The following sections describe the defined requirements, software used for the implementation, implementation process, and configuration steps involved in building the prototype.

4.1 Requirements

The goal of the prototype was to enable real-time monitoring, evaluation, and visualization of students' activity in the sandboxes of the PB177 lab exercises using the OpenSearch Stack. Development began with access to a repository provided by the thesis advisor, which contained an existing OpenSearch Stack configuration designed for deployment on virtual machines. The development was guided by the following requirements, which define the prototype's intended purpose:

1. **Integration of the PB177 lab infrastructure with the OpenSearch Stack**
   The Docker-based infrastructure of the PB177 lab exercises must be integrated into the OpenSearch Stack to enable monitoring of student activity during the lab exercises.

2. **Monitoring and identification of students' activity**
   The prototype must capture every command that students execute in the shell of the attacker container. This includes both commands that create new processes and those that do not, as well as incorrectly entered or failed commands. Additionally, instructors must be able to distinguish which student executed each command. The goal is to provide a complete overview of each student's behavior during a lab exercise.

3. **Reliable operation**
   The logging functionality must be robust and reliable even if a student enters the shell of the attacker container by bypassing

standard startup procedures through alternative methods or changes the user within the container. The system must consistently capture activity regardless of how the attacker container is accessed and what actions are performed within it.

4. **Filtering irrelevant logs**
   The prototype must be able to filter out irrelevant or redundant information and retain only data that is useful for instructors to support effective evaluation and feedback. This includes details such as the executed command, working directory, user name, container name, IP address, and the source from which the command was executed.

5. **Visualization of collected data**
   The collected data must be accessible through OpenSearch Dashboards, enabling instructors to easily explore and analyze students' activity during the lab sessions via a user-friendly graphical interface.

4.2 Software used for the implementation

This section describes the software components used for the implementation of the prototype. These components are described in the following subsections.

Ubuntu server

The prototype was implemented on Ubuntu 22.04 (Jammy Jellyfish), a stable and widely used Linux distribution. Ubuntu provided compatibility with all required software components and served as a reliable environment for development.

Docker Compose

Container management was handled using *Docker Compose*, version 2.29.6. It is configured using the *docker-compose.yml* file, which defines all containers the application uses. This file also enables interaction with the defined containers, including starting, stopping, and checking their status.

Overview of the initial OpenSearch Stack repository

The initial OpenSearch Stack repository included the following files:

- *docker-compose.yml* – Defined all OpenSearch Stack components, including two OpenSearch nodes, OpenSearch Dashboards, and Logstash. Each container was pre-configured with environment variables, port mappings, memory settings, and data volumes. The file also ensured that all components operated within a shared Docker network, which allowed communication between them.
- *opensearch-dashboards.yml* – Specified configuration settings for OpenSearch Dashboards such as hostname, TLS settings, login credentials, and target OpenSearch nodes. It was designed to ensure secure and functional connections between the OpenSearch Dashboard interface and OpenSearch nodes.
- *.env* – Defined environment variables such as the initial administrator password used by both OpenSearch nodes and a domain name. These variables were referenced in the *docker-compose.yml* file to avoid hardcoding sensitive information directly in the configuration.

Additionally, the repository also contained the following directories:

- *pipeline* – This directory holds Logstash pipeline configuration files. In this case, it contained the *test.yml* file, which defines a Logstash pipeline for parsing and forwarding logs. The pipeline was configured to receive input from Beats and output the data to OpenSearch nodes for storage and indexing. The filter section needed to be adjusted to meet the requirements of the prototype.
- *client* – This directory included the *auditbeat.yml* file, which defined the initial configuration for Auditbeat.

4.3 Implementation

This section describes the implementation of the prototype. It is divided into five subsections:

1. **Self-signed certificate for securing OpenSearch Dashboards** – This subsection describes the process of establishing secure HTTPS communication for OpenSearch Dashboards by generating and configuring a self-signed certificate.

2. **Integration of the attacker container with the OpenSearch Stack** – This subsection involves incorporating the attacker container into the OpenSearch Stack and making the necessary configuration adjustments to ensure its proper functionality.

3. **Configuration of Beats** – This subsection focuses on preparing the configuration files for Auditbeat and Filebeat on the host system to eliminate the need for manual configuration within the attacker container.

4. **Installation of Beats inside the container** – This subsection describes the installation of services within the attacker container that are required to collect and forward logs.

5. **Filtering and processing events** – This subsection explains how Logstash receives events from Beats, filters out irrelevant information, transforms the data, and forwards it to OpenSearch for storage and analysis.

Self-signed certificate generation and configuration

The first step of the implementation was to enable secure HTTPS communication for OpenSearch Dashboards. Without HTTPS, sensitive data – including login credentials and dashboard visualizations – would be transmitted in plaintext, posing a security risk. Since this prototype was developed in a local environment, a self-signed TLS certificate was generated using *OpenSSL* [22]. The `openssl` command was used to generate the following files:

- *cert.pem* – The public certificate used by the server to establish a secure TLS connection with clients.

- *privkey.pem* – The private key corresponding to the certificate used to decrypt incoming encrypted data and complete the TLS handshake.

Both files were placed in the `./letsencrypt/live/${FQDN}/` directory on the host system, from which they are mounted into the `opensearch-dashboards` container at their respective destination paths. This is specified in the container's volume configuration, as shown in Listing 11. Although the directory path contains *letsencrypt* in its name, it was used only for convenience with future deployments, as the self-signed TLS certificate can later be replaced with an actual *Let's Encrypt* certificate in a production environment.

```
volumes:
  - "./letsencrypt/live/${FQDN}/cert.pem:/usr/share/
    opensearch-dashboards/config/client-cert.pem"
  - "./letsencrypt/live/${FQDN}/privkey.pem:/usr/share/
    opensearch-dashboards/config/client-cert-key.pem"
```

Listing 11: Mounting certificate files in the `opensearch-dashboards` container

Integration of the attacker container with the OpenSearch Stack

The next step in the implementation involved the integration of the attacker container from the first lab exercise into the OpenSearch Stack. This integration was necessary to ensure that the OpenSearch Stack and the attacker container could operate within the same Docker network. Without running them together, the OpenSearch Stack could not monitor students' activity inside the attacker container. Therefore, launching the attacker container alongside the OpenSearch Stack was crucial for simplifying the development and testing of the prototype.

This integration was achieved by copying the default configuration of the attacker container from the original *docker-compose.yml* file of the first lab exercise into the *docker-compose.yml* file of the OpenSearch Stack. After transferring the configuration, several modifications, highlighted in yellow in Listing 12, were necessary to ensure correct operation of the container:

- **Mounting auditbeat and filebeat configurations:** Additional volumes were mounted into the attacker container:
  - *auditbeat.yml* – the configuration file for Auditbeat.
  - *filebeat.yml* – the configuration file for Filebeat.
  - *system.yml* – the configuration file for the system module used by Filebeat, which enables the collection of logs from Syslog.

  These volumes allow the configuration of Beats to be defined once on the host machine, which is then automatically loaded into the container each time it starts.

- **Enabling privileged mode:** The `privileged: true` command was added to grant extended permissions inside the container, which are necessary for Auditbeat to access the Linux Audit framework.

- **Sharing the host's PID namespace:** The `pid: host` command was added to allow the attacker container to see all processes running on the host. Although this is necessary for proper Auditbeat functionality, it compromises container isolation by allowing Auditbeat to collect events not only from the attacker container but also from the host. Additionally, it creates complications when multiple attacker containers run simultaneously on the same host, as Auditbeat cannot accurately determine the source container of the event. This problem is described in more detail in Section 4.5.

- **Adding new capabilities:** Two additional capabilities were added to the *cap_add* list to support Auditbeat functionality:
  - `AUDIT_CONTROL` – allows the attacker container to enable kernel auditing, manage audit rules, and retrieve auditing status.
  - `AUDIT_READ` – allows the container to read audit logs.

- **Network adjustments:** The attacker container was originally configured to use the `vnet` network. For compatibility with the

OpenSearch Stack, it was changed to the `opensearch-net` network. This adjustment enables direct communication between the attacker container and other components of the stack.

```
attacker:
    container_name: attacker
    hostname: attacker
    image: gitlab.fi.muni.cz:5050/cybersec/infra/images/
    kali/kali:main
    restart: on-failure
    volumes:
      - ./student_home/root:/root
      - user-data:/home
      - ./student_home/xuser/shared:/home/xuser/shared
      # copy auditbeat.yml from host to container
      - ./client/auditbeat.yml:/etc/auditbeat/
        auditbeat.yml
      # copy filebeat.yml from host to container
      - ./client/filebeat.yml:/etc/filebeat/filebeat.yml
      # copy system.yml from host to container
      - ./client/system.yml:/etc/filebeat/modules.d/
        system.yml
    tty: true
    privileged: true
    pid: host
    cap_add:
      - NET_ADMIN
      - NET_RAW
      - SYS_ADMIN
      - DAC_READ_SEARCH
      - AUDIT_CONTROL
      - AUDIT_READ
    ports:
      - "127.0.0.1:14000:13389"
    networks:
      opensearch-net:
```

Listing 12: Modified configuration of the attacker container

Installation of Beats inside the container

Although the Beats configuration files are transferred to the attacker container during startup, the Beats themselves are not included in the

attacker container image and therefore must be installed manually inside the container. The most effective solution would be to include Beats in the attacker container image, ensuring that they are already installed when students start the container.

However, to avoid unnecessary complexity, we decided not to modify the attacker container image, as enhancing the image can affect other sandboxes that use the same image. Instead, I created a shell script that installs and configures all the required services. This script will later be used as a basis for modifying the attacker container image.

This script installs Auditbeat, Filebeat and `rsyslog` [23]. The `rsyslog` is required because Filebeat is configured to collect logs from Syslog. Listing 13 shows the part of the script, which ensures that all commands executed within the container are captured and sent to Syslog, where they are collected by Filebeat.

```
echo "trap 'logger \"PWD: \$PWD CMD: \$BASH_COMMAND\"'
DEBUG" >> /etc/bash.bashrc
source /etc/bash.bashrc
```

Listing 13: Bash configuration for capturing executed commands

Executed commands are captured by setting a `trap` command [24] in the Bash. This `trap` is triggered before each command is executed. It uses the `logger` command [25] to capture the current working directory using the `PWD` variable and the command itself using the `BASH_COMMAND` variable. The `logger` then sends the captured data to Syslog. An example of such data in the Syslog is shown in Listing 14. The `source /etc/bash.bashrc` command allows the changes to take effect immediately in the current Bash session. This configuration enables the capture of all commands – including those that create new processes, those that do not (such as shell built-in commands), and even mistyped or non-existent commands.

```
2025-04-28T14:03:33.556849+02:00 attacker root: PWD: /root
CMD: nmap localhost
```

Listing 14: Example of a captured data in Syslog

After installing the services, the script starts them, using the commands shown in Listing 15. Once running, they monitor, collect, and forward all commands executed within the attacker container.

```
sudo service auditbeat start
rsyslogd
sudo service filebeat start
```

Listing 15: Commands for starting Auditbeat, Filebeat, and `rsyslog`

Configuration of Beats

Since the configuration files for the Beats are mounted at container startup, they must be prepared in advance on the host system. This approach provides centralized control over Auditbeat and Filebeat settings, eliminating the need for manual configuration within the containers.

- **Auditbeat Configuration:** The Auditbeat configuration includes the *auditd* module to monitor process executions. This module contains two rules, presented in Listing 16. These rules enable Auditbeat to monitor system calls for command executions on both 64-bit and 32-bit architectures. However, one limitation of this configuration is its inability to capture all shell activity – particularly commands that are entered into the shell but fail to execute, such as mistyped or non-existent commands, as well as shell built-in commands that do not result in creating a new process.

```
auditbeat.modules:
- module: auditd
  audit_rules: |
        -a exit,always -F arch=b64 -S execve
        -a exit,always -F arch=b32 -S execve
```

Listing 16: Configuration of the Auditbeat auditd module

Additionally, the Auditbeat output is configured to forward all collected events to Logstash, as shown in Listing 17. The `hosts` field specifies the destination host and port, with Logstash listening on port 5044. The `verification_mode: none` disables the verification of the TLS certificate because the prototype was developed locally using a self-signed certificate. Disabling verification allows the services to communicate over HTTPS without requiring a certificate signed by a valid certificate authority (CA), which is acceptable in this case but must be avoided in production environments. Finally, the `protocol: https` setting enables encrypted communication between Auditbeat and Logstash.

```
output.logstash:
  hosts: ["logstash:5044"]
  verification_mode: none
  protocol: "https"
```

Listing 17: Auditbeat output configuration

- **Filebeat configuration:** The Filebeat configuration is shown in Listing 18.
  - The `path` field specifies the directory from which Filebeat loads module configurations. A module must be enabled in order to be loaded.

- The `reload.enabled: true` allows Filebeat to dynamically reload updated configurations without requiring a restart.
- The `reload.period: 10s` defines how often Filebeat checks for changes in the configuration files. Ten seconds is the default value and provides a suitable balance between timely updates and system performance.

```
filebeat.config.modules:
  path: ${path.config}/modules.d/*.yml
  reload.enabled: true
  reload.period: 10s
```

Listing 18: Configuration of Filebeat modules

In this setup, only the *system* module is enabled, as shown in Listing 19. This module is responsible for collecting system logs. It enables Syslog collection, instructing Filebeat to read logs from `/var/log/syslog`.

```
- module: system
  syslog:
    enabled: true
    var.paths: ["/var/log/syslog"]
```

Listing 19: System module configuration

The Filebeat output configuration is identical to the Auditbeat output configuration, meaning that all collected events are also forwarded to Logstash at port 5044, as shown in Listing 17.

Filtering and processing events

After Beats forward events to Logstash, Logstash processes them and filters for relevant data necessary for instructors of PB177. The

Logstash configuration is defined in the *test.yml* file. The structure of this file is divided into three sections: input, filter, and output.

1. **Input:** The input section, shown in Listing 20, is configured to listen for incoming events sent by Beats on port 5044. Once events are received, they are forwarded to the filter section for processing.

```
input {
  beats {
    port => 5044
  }
}
```

Listing 20: Logstash input section

2. **Filter:** The filter section is divided into three parts. The first part is responsible for filtering Filebeat events, the second for filtering Auditbeat events, and the third for keeping only the relevant information.

   - **Filtering Filebeat events:** The configuration of this part is shown in Listing 21. First, Logstash checks whether the received event is from Filebeat. If so, it further verifies whether the event contains the string "CMD", as illustrated in the event shown in Listing 14. This check helps distinguish whether the event was generated by a student, as file `/var/log/syslog` also captures events generated by the system. If the event does not contain the "CMD" field, it is dropped using the `drop{}` command.

     If the event was generated by a student, it is passed to the *Grok* filter, which parses the *message* field and extracts individual values into corresponding fields. After parsing, the *date* filter converts the extracted *timestamp* from a string into a proper timestamp format. Then, a *mutate* filter creates two additional fields: *cmd_source*, which is set to the static

value "logger" to indicate the event was captured by `logger`, and *ip_address*, which stores the IP address extracted from the received event.

However, the *ip_address* field captures the attacker container's network IP address, which is the same for all students. Therefore, an additional mechanism is required to uniquely identify individual students.

```
if [agent][type] == "filebeat" {
  if "CMD" in [message] {
    grok {
      match => { "message" => "%{TIMESTAMP_ISO8601:
      timestamp}\s%{GREEDYDATA:container_name}
      \s%{GREEDYDATA:user_name}:
      \sPWD\s%{GREEDYDATA:working_directory}
      \sCMD:\s%{GREEDYDATA:cmd}" } }
    date {
      match => ["timestamp", "yyyy-MM-dd'T'HH:mm:ss.
      SSSSSSZZ"]
      target => "@timestamp" }
    mutate {
      add_field => {
        "cmd_source" => "logger"
        "ip_address" => "%{[hostname][ip]}" } }
  }
  else { drop {} }
}
```

Listing 21: Filtering of Filebeat events in Logstash

- **Filtering Auditbeat events:** The configuration for filtering Auditbeat events is presented in Listing 22. Logstash first verifies whether the event is from Auditbeat. If so, it checks whether the event contains the *process* field, as events generated by a student always include this field. However, it should be noted that events generated by the system can also contain the *process* field and are therefore kept as well.

This is not sufficient solution, as those system-generated events are not relevant for the purpose of this prototype. After the check, the *mutate* filter creates six fields, storing values such as the executed command, working directory, user name, container name, and IP address received from the event. The *cmd_source* field is set to the static value "auditd" to indicate that the event was captured by Auditd. If no *process* field is present in the event, it is dropped.

```
if [agent][type] == "auditbeat" {
  if [process] {
    mutate {
      add_field => {
        "cmd" => "%{[process][title]}"
        "cmd_source" => "auditd"
        "working_directory" => "%{[process][working_
                                directory]}"
        "container_name" => "%{[hostname][name]}"
        "user_name" => "%{[user][name]}"
        "ip_address" => "%{[hostname][ip]}" } }
  }
  else { drop {} }
}
```

Listing 22: Filtering of Auditbeat events in Logstash

Auditbeat also includes filtering in the *auditbeat.yml* file, as shown in Listing 23. It uses the `processors` utility to drop all events generated by the `logger` command since Filebeat already captures those events and would otherwise be duplicated.

```yaml
processors:
  - drop_event:
      when:
        and:
          - equals:
              agent.type: "auditbeat"
          - equals:
              process.name: "logger"
```

Listing 23: Filtering of `logger` events in *auditbeat.yml*

- **Whitelisting fields:** While the previous two parts focused on creating relevant fields, storing their corresponding values, and dropping irrelevant events, this part modifies the already processed events by removing all unnecessary fields. The goal is to make the events clearer by keeping only the newly created fields. The configuration of this part is presented in Listing 24. It uses the *prune* filter to keep only the fields specified in the *whitelist_names* list and remove all others from the processed events. Once this part is completed, the entire filtering process is finished.

```
prune {
  whitelist_names => ["@timestamp", "cmd", "cmd_source",
  "working_directory", "container_name", "user_name",
  "host_ip"]
}
```

Listing 24: Whitelisting of fields in Logstash

3. **Output:** The output section is responsible for forwarding the processed events to OpenSearch. Its configuration is shown in Listing 25.

- The *hosts* field specifies the destination node for storing the events. In this case, the processed events are sent to the OpenSearch node named `opensearch-node1`, which listens on port 9200 over HTTPS.
- The *index* field defines the index where the documents will be stored in OpenSearch. The pattern `opensearch-logstash-docker-%+YYYY.MM.dd` dynamically creates a new index for each day based on the current date.
- The *user* and *password* fields provide the credentials required for authentication with OpenSearch.
- The *ssl* field, despite its name referencing the older SSL protocol, enables TLS encryption for secure communication between Logstash and OpenSearch nodes.
- Similarly, the *ssl_certificate_verification* field, despite its name referencing the SSL protocol, turns off Logstash's verification of the TLS certificate presented by OpenSearch nodes.

```
output {
  opensearch {
     hosts => ["https://opensearch-node1:9200"]
     index => "opensearch-logstash-docker-%{+YYYY.MM.dd}"
     user => "admin"
     password => "<REDACTED>"
     ssl => true
     ssl_certificate_verification => false
  }
}
```

Listing 25: Output section of Logstash

4.4 Visualization

The final part of the implementation involved the visualization of processed events, which is done using OpenSearch Dashboards. OpenSearch

Dashboards is configured via the *opensearch_dashboards.yml* file, shown in Listing 26.

```
server.host: "0.0.0.0"
server.name: "localhost"
server.ssl.enabled: true
server.ssl.certificate: /usr/share/opensearch-dashboards/
                        config/client-cert.pem
server.ssl.key: /usr/share/opensearch-dashboards/
                config/client-cert-key.pem
opensearch.hosts: ["https://opensearch-node1:9200",
                   "https://opensearch-node2:9200"]
opensearch.ssl.verificationMode: none
opensearch.username: "admin"
opensearch.password: "<REDACTED>"
```

Listing 26: Configuration of *opensearch_dashboards.yml*

Below is a breakdown of the configuration used in this file. Although several configuration parameters contain "SSL" in their names, it is important to note that they use the TLS protocol.

- *server.host*: Set to `"0.0.0.0"`, allowing OpenSearch Dashboards to accept connections on all network interfaces.
- *server.name*: Defines the server's name, which is set in this case to `"localhost"`.
- *server.ssl.enabled*: Set to `true`, enabling HTTPS for secure access using TLS.
- *server.ssl.certificate* and *server.ssl.key*: These fields specify the paths to the TLS certificate and its private key used to establish secure HTTPS connections through TLS. This certificate is different from the one used by Logstash.
- *opensearch.hosts*: Lists the addresses of the OpenSearch nodes: `opensearch-node1` and `opensearch-node2`, which are used by OpenSearch Dashboards to retrieve and display data.

- *opensearch.ssl.verificationMode*: Set to `none`, disabling client-side verification of the TLS certificate.
- *opensearch.username* and *opensearch.password*: Provide the login credentials to authenticate to OpenSearch Dashboards.

To access OpenSearch Dashboards, the user opens a web browser and navigates to `https://localhost:5602`. After logging in, users can explore received events. Figure 4.1 shows an example of a received event displayed in OpenSearch Dashboards.

	@timestamp	Apr 28, 2025 @ 14:03:33.556
t	_id	iI1IfJYBkeh1-uZgNVvC
t	_index	opensearch-logstash-docker-2025.04.28
#	_score	-
t	_type	-
?	cmd	nmap localhost
?	cmd_source	logger
?	container_name	attacker
?	user_name	root
?	working_directory	/root

Figure 4.1: Example of an event in OpenSearch Dashboards

4.5 Limitations

While the prototype achieves its core objectives – monitoring, filtering, and visualizing students' activity inside the attacker container –

several known limitations remain. These were identified during the implementation but were not addressed, either because they exceeded the prototype's scope or were deferred as suggestions for future enhancement.

1. **Unfiltered logger messages in Syslog** – All command executions captured by the `logger` utility are sent directly to `/var/log/syslog`. However, as mentioned in Section 4.3, Syslog also receives events not captured by the `logger` – specifically, system-generated events. Although a "CMD" string was added to identify events generated by students, this approach is insufficient, as it does not accurately determine whether an event was generated by a student or by the system. For example, if a student changes the user within the attacker container, this action triggers multiple system commands that are sent to Syslog, which also contain the "CMD" string, even though they were not directly executed by the student. As a result, these irrelevant events are logged and displayed in OpenSearch Dashboards, which instructors do not want to see. A more robust solution would be to redirect `logger` output to a dedicated log file, thereby avoiding the mixing of student-generated commands with system-generated messages.

2. **Missing identification of student machines** – The prototype logs the IP address of the attacker container. As a result, instructors cannot determine which student executed a particular command since all logs in OpenSearch Dashboards show the same container IP address across all host machines. Future solutions should implement a mechanism for logging the host machine's IP address, which is unique to each machine. Alternatively, the student's UČO can be passed as an environment variable when starting the attacker container. This value can then be included in the events sent by Auditbeat and Filebeat. These solutions would allow instructors to reliably identify which student executed each command.

3. **Lack of logging functionality in non-Bash environments** – The `trap` command described in Section 4.3 is appended only to `/etc/bash.bashrc`, meaning it works only in Bash. If a student

uses an alternative shell, such as `sh`, the trap is not triggered, and executed commands will not be logged by `logger`. As a result, only commands that create new processes are captured by *Auditd*, while commands that do not create new processes and incorrectly entered commands remain unlogged. The future solution should guarantee proper functionality of `logger` in shell environments other than Bash.

4. **Inability to distinguish between multiple attacker containers** – Auditbeat can correctly determine the source container of an event only when a single attacker container is running on the host. When multiple attacker containers with different names run simultaneously on the same host, Auditbeat cannot accurately associate events with the correct container. This issue arises from the use of the `pid: host` command in the *docker-compose.yml* file of the OpenSearch Stack, as discussed in Section 4.3. To ensure that instructors can identify the source container of each event in multi-container setups, a more reliable mechanism for container identification must be implemented.

5. **Incomplete filtering of multi-event commands** – Auditbeat filtering does not work as expected for certain commands due to limitations discovered during the implementation of Auditbeat filtering in Logstash. While most commands logged by Auditbeat generate only a single event, which can be filtered correctly, some commands – such as `nmap` or `sqlmap` – produce multiple events. In such cases, it is necessary to determine from which event the relevant information will be extracted and discard the rest.

   A partial solution was implemented using the `processors` utility in the *auditbeat.yml* file, as shown in Listing 27. This configuration drops all events generated by `nmap` except the one where the *auditd.summary.object.primary* field equals "/usr/bin/nmap". From this remaining event, relevant information is extracted and forwarded to instructors. However, since this solution only works for `nmap`, it is necessary to generalize the filtering to support all commands that generate multiple events.

```
processors:
  - add_host_metadata: ~
  - add_docker_metadata: ~
  - drop_event:
      when:
        and:
          - contains:
              process.title: "nmap"
          - not:
              equals:
                auditd.summary.object.primary: "/usr/bin/
                nmap"
```

Listing 27: Auditbeat filtering of multiple events generated by `nmap`

6. **Separation of the attacker container and OpenSearch Stack** – One of the initial steps in the implementation was integrating the attacker container with the OpenSearch Stack. Although this integration simplified the development and testing of the prototype, it does not represent the intended target state. The goal is to run attacker containers on separate machines operated by individual students, with a central virtual machine hosting the OpenSearch Stack. Future work should focus on separating these components to enable this distributed setup.

7. **Missing CA-signed certificates** – In the Logstash and OpenSearch Dashboards configuration files, TLS certificate verification was disabled to allow HTTPS communication using a self-signed certificate during local development. This practice is acceptable in a development environment but must be properly addressed by using CA-signed certificates for production use.

These limitations offer an overview of future improvements that must be addressed before deploying the prototype to a production environment.

5 Conclusion

This thesis explored the potential of the INJECT Exercise Platform (IXP) for conducting PB177 Cyber Attacks course lab exercises and developed a prototype for real-time monitoring of students' activity in these exercises.

The first significant contribution of this thesis was the adaptation of the first PB177 lab exercise to IXP's exercise definition format. This process revealed several missing features that limited the platform's suitability for conducting the PB177 lab exercises. These features – such as the inability to submit free-form answers or include notes beneath questions – were reported to the IXP development team and led to partial enhancements to IXP's exercise definition.

The second contribution was the implementation of a prototype for monitoring students' activity in the first PB177 lab exercise. The prototype leveraged the OpenSearch Stack to collect, process, and visualize commands executed inside the attacker container used in the lab exercise. The development was guided by a set of defined requirements focusing on reliable operation, meaningful data collection, and clear visualization for instructors. Although the prototype successfully demonstrated the feasibility of capturing students' interactions with the attacker container during the lab exercise, several limitations were identified – particularly the inaccurate filtering of system-generated data, the inability to support multiple containers, and the lack of support for non-Bash environments.

Despite these limitations, the developed prototype serves as a solid foundation for future improvements. It presents a viable approach to increasing observability in the PB177 lab exercises. Future enhancements can focus on addressing the identified limitations, improving the robustness of the prototype, and extending its applicability beyond PB177.

Ultimately, this thesis demonstrates the feasibility of integrating existing PB177 lab exercises into IXP. Such integration can enhance their educational value by providing students with a more structured and interactive learning environment while equipping instructors with a comprehensive tool for effectively monitoring and evaluating students' activity during the lab exercises.

A Content of the attachments

The attachment of this thesis, available in the Information System of Masaryk University (`https://is.muni.cz/auth/th/f1gy8/`), contains a single ZIP archive. This archive consists of the following two directories:

- `definitions` – a directory containing the IXP exercise definition for the first lab exercise,
- `opensearch` – the root directory for all configuration files and scripts related to the developed prototype,
- `README.md` – a file providing details about the attachment.

Bibliography

1. ALRABAEE, Saed; AL-KFAIRY, Mousa; BARKA, Ezedin. Efforts and Suggestions for Improving Cybersecurity Education. In: *2022 IEEE Global Engineering Education Conference (EDUCON)*. 2022, pp. 1161–1168. Available from DOI: 10.1109/EDUCON52537.2022.9766653.
2. MORALES-GONZALEZ, Christopher; HARPER, Matthew; RAYAVARAM, Pranathi; NARAIN, Sashank; FU, Xinwen. Enhancing Cybersecurity Education using Scoring Engines: A Practical Approach to Hands-On Learning and Feedback. In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. Pittsburgh, PA, USA: Association for Computing Machinery, 2025, pp. 784–790. SIGCSE TS 2025. Available from DOI: 10.1145/3641554.3701905.
3. *PB177 Cyber Attacks – Course Information* [online]. Brno: Masaryk University, 2024 [visited on 2025-03-20]. Available from: https://is.muni.cz/predmet/fi/podzim2024/PB177.
4. ŠVÁBENSKÝ, Valdemar; VYKOPAL, Jan; HORÁK, Martin; HOFBAUER, Martin; ČELEDA, Pavel. From Paper to Platform: Evolution of a Novel Learning Environment for Tabletop Exercises. In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. Milan, Italy: Association for Computing Machinery, 2024, pp. 213–219. ITiCSE 2024. Available from DOI: 10.1145/3649217.3653639.
5. OPENSEARCH. *OpenSearch* [online]. [visited on 2025-05-17]. Available from: https://opensearch.org/.
6. CONE, Matt. *What is Markdown?* [online]. [visited on 2025-05-19]. Available from: https://www.markdownguide.org/getting-started/.
7. RED HAT. *What is Podman?* [online]. [visited on 2025-05-17]. Available from: https://docs.podman.io/en/latest/.
8. DOCKER. *Docker* [online]. [visited on 2025-05-22]. Available from: https://www.docker.com/.

9. YAML LANGUAGE DEVELOPMENT TEAM. *YAML Ain't Markup Language* [online]. 2021. [visited on 2025-03-31]. Available from: https://yaml.org/spec/1.2.2/.

10. INJECT. *INJECT Process* [online]. [visited on 2025-04-06]. Available from: https://docs.inject.muni.cz/INJECT_process/intro/overview/.

11. ELASTIC. *Beats* [online]. [visited on 2025-04-20]. Available from: https://www.elastic.co/beats.

12. ELASTIC. *Filebeat overview* [online]. [visited on 2025-04-01]. Available from: https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html.

13. ARCH LINUX. *Audit framework* [online]. [visited on 2025-04-01]. Available from: https://wiki.archlinux.org/title/Audit_framework.

14. ELASTIC. *Auditbeat overview* [online]. [visited on 2025-04-01]. Available from: https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-overview.html.

15. OPENSEARCH. *Logstash* [online]. [visited on 2025-04-01]. Available from: https://opensearch.org/docs/latest/tools/logstash/index/.

16. OPENSEARCH. *Getting started* [online]. [visited on 2025-04-01]. Available from: https://opensearch.org/docs/latest/getting-started/.

17. OPENSEARCH. *Introduction to OpenSearch* [online]. [visited on 2025-04-02]. Available from: https://opensearch.org/docs/latest/tuning-your-cluster/.

18. OPENSEARCH. *Getting started* [online]. [visited on 2025-04-19]. Available from: https://docs.opensearch.org/docs/latest/getting-started/intro/.

19. OPENSEARCH. *OpenSearch Dashboards* [online]. [visited on 2025-04-02]. Available from: https://opensearch.org/docs/latest/dashboards/.

20. NMAP ORG. *Nmap* [online]. [visited on 2025-04-06]. Available from: https://nmap.org/.

21. KALI ORG. *Dirb* [online]. [visited on 2025-04-06]. Available from: https://www.kali.org/tools/dirb/.

22. SSL DRAGON. *What Is OpenSSL? How Does OpenSSL Work?* [online]. [visited on 2025-05-14]. Available from: https://www.ssldragon.com/blog/what-is-openssl/#what-is-openssl.

23. RSYSLOG. *rsyslog* [online]. [visited on 2025-05-18]. Available from: https://www.rsyslog.com/doc/index.html.

24. LINUX JOURNAL. *The Bash Trap Command* [online]. [visited on 2025-04-26]. Available from: https://www.linuxjournal.com/content/bash-trap-command.

25. IBM. *logger* [online]. [visited on 2025-04-26]. Available from: https://www.ibm.com/docs/en/aix/7.2?topic=l-logger-command.