UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
UNDERGRADUATE SCHOOL

BACHELOR THESIS

By

DUONG Tuan Minh

BI11-172

Information and Communication Technology

Title:

Improve the security of Virtual Resources and Labs deployment for student hands-on trainings

External supervisor: Assoc Prof. CHHEL Fabien - ESEO

Grande École d'Ingénieurs

Internal supervisor: Dr. NGUYEN Minh Huong - ICT Lab

Hanoi, September 2023

Acknowledgements

I would like to acknowledge everyone who contributed to my internship experience, as this thesis would not have been possible without their assistance.

To begin with, I would love to express my deepest appreciation to Associate Professor CAMP Olivier, CHHEL Fabien, and JAMET François at ESEO Grandes Écoles d'Ingénieurs. They not only provided me with the opportunity to be a part of this project but also generously invested their valuable time to mentor and guide me through the internship's challenges.

Additionally, I would like to take this opportunity to thank Dr. Nguyen Minh Huong for the invaluable advice and insights she offered me during my internships as well as during her lectures throughout my time at the university.

Last but not least, I want to sincerely thank my family and friends for their unconditional love and support throughout my journey, especially my three years at USTH.

(DUONG Tuan Minh)
Hanoi, September 2023

Contents

List of Acronyms i

List of Figures ii

Abstract iii

1 Introduction 1

1.1 Overview . . . 1

1.2 Objectives . . . 2

1.3 Thesis Structure . . . 2

2 Theoritical Background 3

2.1 GitOps Philosophy . . . 3

2.1.1 GitLab . . . 3

2.1.2 Ansible . . . 4

2.2 Virtualization Technology . . . 4

2.2.1 Virtual Machine . . . 4

2.2.2 Container . . . 5

2.3 Security Compliance Scan . . . 5

2.3.1 SCAP Content . . . 5

2.3.2 OpenSCAP . . . 5

2.3.3 SCAP Security Guide . . . 6

2.4 Wazuh Monitoring System . . . 6

3 Methodology & Implementation 7

3.1 Environment Setup . . . 7

3.1.1 Management VMs . . . 7

3.1.2 Service VMs . . . 8

3.2 Methodology . . . 9

3.3 OpenSCAP Integration . . . 10

3.4 PenTest Lab Deployment . . . 11

3.4.1 Certificates Generation . . . 12

3.4.2 VPN Server Configuration . . . 13

3.4.3 Clients Generation . . . 14

3.4.4 Backend Server Configuration . . . 15

3.5 Wazuh Monitoring Setup 16

4 Results **18**

4.1 Git Repositories 18

4.2 OpenSCAP Compliance Scan 19

4.3 PenTest Lab Usage 21

4.4 Wazuh Monitoring 23

5 Discussion **25**

5.1 Conclusion 25

5.2 Discussion 25

5.3 Future Works 26

References **28**

List of Acronyms

CA	Certificate Authority.
CI/CD	Continuous Integration and Continuos Delivery.
CVEs	Common Vulnerabilities and Exposures.
DH	Diffie-Hellman.
HMAC	Hash-based Message Authentication Codes.
IaC	Infrastructure as Code.
OS	Operating System.
OWASP	Open Worldwide Application Security Project.
PKI	Public Key Infrastructure.
RSA	Rivest–Shamir–Adleman.
SCAP	Security Content Automation Protocol.
SSG	SCAP Security Guide.
SSH	Secure Shell.
TLS	Transport Layer Security.
UFW	Uncomplicated Firewall.
VM	Virtual Machine.
VPN	Virtual Private Network.

List of Figures

3.1 Web Interface of local GitLab Server 8
3.2 Local environment mirroring ESEO infrastructure 9
3.3 Packets from VPN clients being masqueraded before reaching backend server . . . 14

4.1 Git Repositories on GitLab Server 18
4.2 Git Repositories on GitHub 19
4.3 OpenSCAP Integration Pipeline 19
4.4 OpenSCAP versions automatically updated using CI jobs 20
4.5 SCAP security guide versions automatically updated using CI jobs 20
4.6 Example OpenSCAP compliance scan report 21
4.7 PenTest Lab Deployment Pipeline 21
4.8 Example connection to VPN server 22
4.9 Example SSH connection to the "attacker" container as student 22
4.10 Example exploit of shellshock vulnerability 22
4.11 WebGoat web interface 23
4.12 Wazuh Deployment Pipeline 23
4.13 Wazuh Dashboard 24

Abstract

This collaborative project with ESEO Grandes Écoles d'Ingénieurs aimed to improve both the security and educational aspects of the institution's internal cloud infrastructure. In the beginning phase, we integrated OpenSCAP, a security compliance scanning tool, into the existing deployment pipelines with the intention of performing security audits during the provisioning of Virtual Machine (VM). This enhancement ensures that ESEO can provide computing resources to facilitate their educational activities more securely. As the project evolved, we added more complexity with the deployment of a security training environment called PenTest Lab. This lab incorporates two components, including an OpenVPN server for secure access control and Docker containers housing intentionally vulnerable targets, offering a controlled space for students to practice their penetration testing skills. Additionally, to heighten the security of this PenTest Lab, Wazuh - a well-known open-source real-time monitoring system - was also deployed. To ensure the efficient management of these systems, our project followed the GitOps operational framework, leveraging GitLab CI/CD pipelines and Ansible automation. Our projects resulted in the creation of three Git repositories, which act as an automated solution to address the mentioned tasks. These repositories are being shared publicly on GitHub since they can be valuable to any party facing similar challenges.

Keywords: *Cloud, Automation, GitOps, Security Compliance, PenTest Lab, Monitoring System*

Section 1

Introduction

1.1 Overview

ESEO Grandes Écoles d'Ingénieurs, a French engineering school, is the hosting entity for this internship project. They employ **OpenStack**, an **internal cloud infrastructure** running on ESXi servers, to support their educational activities. However, their system currently lacks the capability to conduct comprehensive security audits when provisioning new Virtual Machine. Therefore, in the initial phase of the internship, we will address this problem by integrating OpenSCAP - a **security compliance scanning tool** into the existing deployment pipelines. This integration presents two distinct challenges. Firstly, the intern must establish a localized network of Virtual Machines to emulate a portion of ESEO's infrastructure due to security constraints limiting direct access to the institution's internal system. Secondly, as of this writing, there exists no method to install OpenSCAP from the official package repository for Debian 11 - ESEO's cloud primary operating system. Consequently, the utilization of OpenSCAP mandates manual compilation and management of the latest version instead of leveraging package management tools.

As the project progressed, an additional layer of complexity was introduced by incorporating the **deployment of a PenTest Lab** on ESEO's cloud. The PenTest Lab is designed to serve as a controlled environment, offering multiple intentionally vulnerable targets for cybersecurity students at ESEO to train their penetration testing skills. This specialized lab necessitates the deployment of two Virtual Machines, each assigned distinct roles: managing access control and hosting lab challenges. First of all, to ensure restricted access and confine any intrusive activities solely to the designated sandbox environment, an OpenVPN server must be established. Within the second VM, all the vulnerable targets for the lab would be hosted inside Docker containers.

Lastly, to improve the security of the PenTest Lab, we opted to introduce a **monitoring system powered by Wazuh**. Because of Wazuh's agent-based architecture, installing Wazuh agents on the two VMs responsible for the PenTest Lab is imperative. These agents are responsible for real-time data collection. Additionally, we established a third Virtual Machine dedicated to hosting the Wazuh central components, facilitating the analysis of data gathered from these agents.

All tasks outlined above are subject to a significant constraint, which involves the utilization of GitLab CI/CD pipeline and Ansible in **alignment with the GitOps operational framework** to

automate infrastructure management proficiently. ESEO has embraced GitOps philosophy for governing their virtual resources. Consequently, adopting this methodology is essential to ensure that the outcomes of our work remain relevant and directly applicable to our host institution.

Given that the GitOps methodology requires the infrastructure deployment and management processes to be represented as code or configuration files, our efforts will extend beyond the immediate benefit to ESEO. They have the potential to assist other individuals and organizations in efficiently deploying similar infrastructures or addressing similar challenges with minimal modifications required.

1.2 Objectives

The primary goal of this internship project is to create three Git repositories in accordance with the principles of GitOps, enabling the automation of the tasks previously outlined. These repositories will serve distinct purposes:

- The first repository will be dedicated to automating the integration of OpenSCAP, handling package compilation, version tracking, and the execution of security audit scans.
- The second repository will focus on the deployment of the PenTest Lab, encompassing the setup of the OpenVPN server and Docker containers to create a controlled environment.
- The third repository will be designed to streamline the deployment of Wazuh. Notably, this repository is separated from the second one to expand its utility to other systems, extending its applicability beyond monitoring the PenTest Lab.

1.3 Thesis Structure

Our thesis is organized into five sections as follows:

- Section 1: Establishes the context and outlines the objectives of this thesis
- Section 2: Provides theoretical background of the tools and concepts pivotal to this project
- Section 3: Describes in detail the work process and methods implemented in our work
- Section 4: Presents the outcomes of the internship
- Section 5: Concludes the work and discusses future improvement for the project

Section 2

Theoritical Background

2.1 GitOps Philosophy

DevOps is a methodology that combines various practices and tools. It focuses on improving the collaboration between development and operation teams in order to shorten the time it takes to deliver applications. GitOps evolved from DevOps by inheriting its best practices for software development, such as version control, collaborative workflows, and CI/CD, and applying them to automate infrastructure management [3]. GitOps is a framework that relies on Git repositories as a control mechanism and single source of truth to deliver Infrastructure as Code (IaC). In other words, it is the practice of using Git and CI/CD to store configurations and automatically deploy modifications such as creating, updating, or deleting infrastructure [5].

2.1.1 GitLab

GitLab is a platform that provides a set of tools that facilitate the integration of DevOps methodologies into software development. GitLab includes functions such as code repository hosting, version control, CI/CD processes, etc. While GitHub is a more well-known alternative, GitLab is more beneficial for our project due to its integrated CI/CD capabilities and straightforward self-hosting options [6].

Continuous Integration and Continuos Delivery (CI/CD) is a software development method that focuses on continuously developing applications, integrating changes into a shared repository, testing those changes, and delivering them to production. GitLab CI/CD is a built-in feature of GitLab for automatically implementing CI/CD pipelines [4]. In our project, we leveraged GitLab CI/CD to automate the deployment and management of our infrastructure, adhering to the principles of GitOps. This involved representing our infrastructure as code and configuration files within GitLab CI/CD pipelines. By adopting this approach, we ensured that our infrastructure management remained consistent, version-controlled, and reproducible.

To better understand our workflow, it is valuable to review some key terms related to GitLab CI/CD. “Pipelines” are the top-level component of CI/CD processes, comprising a series of jobs and stages. GitLab jobs are essentially tasks that we define, such as compiling a tool or executing a script. These jobs can be grouped into different stages, which define when jobs should be

executed during the pipeline. It's important to note that GitLab jobs are executed on a separate machine from the one hosting the primary GitLab platform. The machine hosting this platform for managing code repositories and tools is referred to as GitLab Server. On the other hand, the machine responsible for executing the defined jobs is known as the GitLab Runner [4]. In this project, we established a local GitLab Server and GitLab Runner. This setup enabled us to automatically trigger and execute three distinct pipelines, each composed of the specific jobs we designed for infrastructure management and automation processes.

2.1.2 Ansible

Ansible is an open-source tool that automates the management and configuration of IT systems. It operates on a client-server architecture, where the controlling machine, often referred to as the "Ansible control node," communicates with remote target machines using OpenSSH — a tool for remote login with the Secure Shell (SSH) protocol [2]. Ansible leverages a "push" approach, where the control node sends automation tasks to the target machines, enabling a centralized and agentless control method. This approach simplifies the initial configuration and makes Ansible stand out from other agent-based tools because it does not require installing and upgrading remote daemons.

The automation workflows in Ansible are defined using "playbooks," which are written in human-readable YAML format and specify the desired state of systems and the series of tasks needed to reach that state. To further enhance code reusability and maintainability, Ansible introduces "roles." Roles are reusable sets of tasks and variables organized into a structured directory, making applying automation across different infrastructures easier [1].

In our setup, the GitLab Runner acted as the control node to execute different playbooks to configure the target machines.

2.2 Virtualization Technology

2.2.1 Virtual Machine

A Virtual Machine (VM) is a software emulation of a physical computer, such as a laptop or server, running an Operating System (OS) and applications just like a physical machine. The difference is that VMs are virtual or software-based computers existing only as code, while physical machines are made of hardware, which is tangible. VMs are created and managed by a hypervisor, a virtualization technology that allows multiple VMs to coexist on a single physical server. Each VM operates independently in an isolated OS and has a designated amount of resources borrowed from a physical host computer, including CPU, memory, storage, etc [12].

2.2.2 Container

Containerization is a lightweight form of virtualization that enables the packaging of application source code with all the required dependencies to create isolated environments called containers that run consistently on any infrastructure. A container is a sandboxed process isolated from other processes running on the host machine [11]. The idea of containerization and process isolation has a long history. However, the widespread adoption of this technology gained significant momentum in 2013 when the open-source Docker Engine was introduced as an industry standard [13].

Docker is a popular containerization platform that simplifies the creation, deployment, and management of containers. A Docker image is a standalone, executable package that includes everything needed to run the software, including the code, runtime environment, system tools, libraries, and settings. These images are created from a set of instructions defined in a Dockerfile and can be easily shared and deployed. Docker containers are instances of Docker images that can be executed on a host system.

Virtual Machines (VMs) and Docker containers serve similar purposes in terms of isolating and running applications, but they differ significantly in their architecture and resource utilization. VMs include a full operating system, resulting in a more substantial resource overhead compared to containers. Containers, on the other hand, share the host OS kernel, making them lightweight and efficient in terms of resource usage.

VMs are better suited for running multiple applications with different OS requirements on a single physical machine, providing strong isolation between VMs. On the other hand, containers are ideal for microservices architectures and application portability due to their lightweight nature and fast startup times.

2.3 Security Compliance Scan

2.3.1 SCAP Content

Security compliance scanning involves using specialized tools to assess whether a system aligns with predefined security policies or standards. These policies, referred to as "SCAP content," dictate the required system configurations and the specific checks to be performed. The Security Content Automation Protocol (SCAP) is a format maintained by the USA's National Institute of Standards and Technology (NIST). Its primary purpose is to establish a standardized approach to maintaining system security.

2.3.2 OpenSCAP

A SCAP scanner is an application that scans and evaluates whether a system is compliant with SCAP security policies. It iteratively goes through the rules outlined in the policy and reports

whether each one is fulfilled. If all checks pass successfully, the system complies with the security policy [7].

OpenSCAP is a popular open-source SCAP scanner that allows users to assess a system's security configuration settings and detect signs of compromise. It accomplishes this by applying rules derived from security policies, enabling comprehensive evaluations of system security. To utilize OpenSCAP, providing security policies in SCAP format is essential. There are many providers of SCAP content, and in this project, we used SCAP content provided by the SCAP Security Guide (SSG) project [8].

2.3.3 SCAP Security Guide

SCAP Security Guide (SSG) is an open-source project that comprises security policies presented in the form of SCAP documents. This project transforms security guidances recommended by respected authorities into a machine-readable format, which can then be used by OpenSCAP to audit your system. These policies span a wide range of computer security domains and offer best-practice solutions. The guide also includes rules with detailed descriptions and provides proven remediation scripts designed for specific target systems. By employing the SCAP Security Guide in conjunction with OpenSCAP, we manage to automate system audits efficiently [9].

2.4 Wazuh Monitoring System

Wazuh is an open-source security monitoring platform that helps organizations detect, analyze, and respond to security threats and incidents in real-time. Wazuh comprises the agent and three central components: the Wazuh server, the Wazuh indexer, and the Wazuh dashboard. The Wazuh indexer is a full-text search and analytics engine that is responsible for indexing and storing alerts produced by the Wazuh server. The Wazuh server, on the other hand, evaluates the data it obtains from the agents, processing it via decoders and rule sets. It leverages threat intelligence to seek out recognizable indicators of security compromise (IOCs). Furthermore, it plays a crucial role in agent management by facilitating remote configuration and upgrades as needed. As for the Wazuh dashboard, it functions as a web-based interface for visualizing and analyzing data as well as for managing Wazuh configuration and monitoring its status. Lastly, Wazuh agents are software installed on various endpoints like laptops, desktops, servers, cloud instances, or virtual machines, offering a comprehensive range of capabilities spanning threat prevention, detection, and response [10].

Section 3

Methodology & Implementation

3.1 Environment Setup

To replicate ESEO's cloud environment, we needed two distinct categories of Virtual Machine: management VMs and service VMs. On one hand, management VMs host the GitLab Server and GitLab Runner and are intended to be managed by a system administrator. These VMs store infrastructure definitions and use the GitLab CI/CD pipeline to deploy changes on the service VMs. On the other hand, service VMs represent Virtual Machines that are hosted on ESEO's cloud. These VMs must be securely configured as they are made available to instructors and students for educational activities.

3.1.1 Management VMs

The management VMs are replicas of the ESEO GitLab Server and Runner. Since the priority of this stage is to promptly deploy a local environment to establish a proof of concept for the internship, we took advantage of two open-source GitHub repositories containing essential scripts for automating these tasks: https://github.com/rgl/gitlab-vagrant and https://github.com/rgl/gitlab-ci-vagrant. The remaining challenges revolve around researching the underlying technologies, specifically Vagrant, making requisite adjustments, and executing the VM deployments.

Vagrant is a versatile tool designed for the creation and management of Virtual Machine environments with a strong emphasis on automation. It achieves this by provisioning VMs on top of existing virtualization platforms, such as VirtualBox or VMWare. Subsequently, Vagrant handles the installation and configuration of essential software components on these VMs. This entire process is orchestrated through a configuration file called the Vagrantfile, which encompasses critical details like hardware specifications, the selected Operating System, and provisioning scripts. The process can then be executed seamlessly through a straightforward command: `vagrant up`.

In this project, we opted to set up our GitLab Server and Runner using VirtualBox, as it is not only highly recommended but also well-documented within the Vagrant documentation. Following the instructions in the GitHub projects mentioned above, with minor adjustments to the network configurations, we successfully instantiated two Virtual Machines. These VMs are based on the Ubuntu 22.04 Operating System and have been respectively equipped with the necessary software

to operate as a GitLab Server and GitLab Runner. Notably, the GitLab Server's web interface, as depicted in Figure 3.1, can be conveniently accessed via our local web browser by navigating to https://gitlab.example.com.

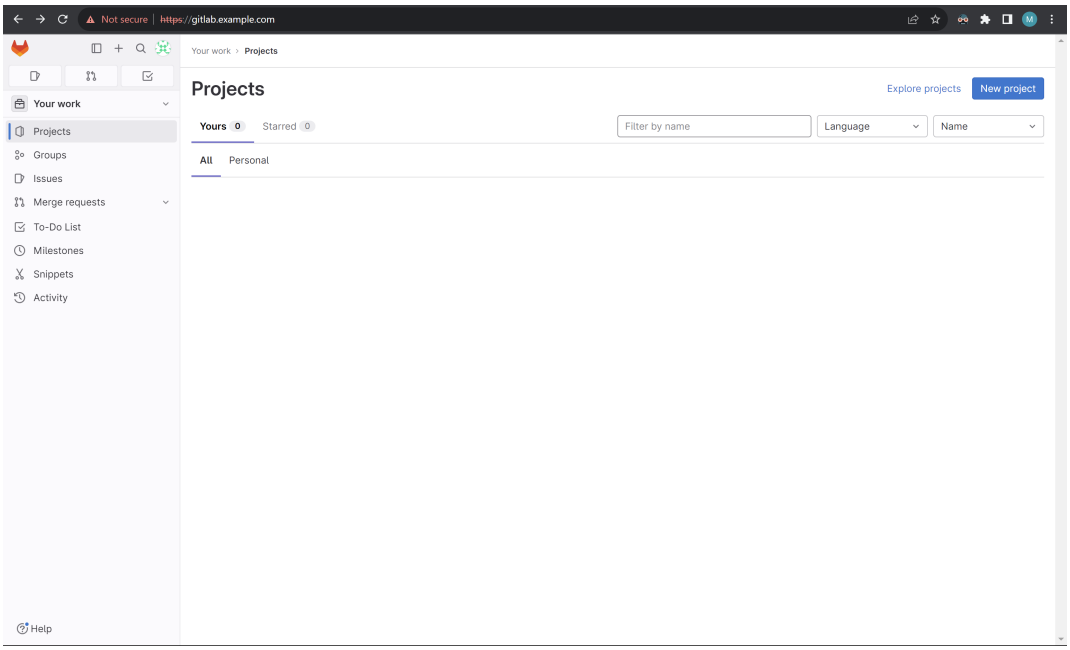

Figure 3.1 – Web Interface of local GitLab Server

3.1.2 Service VMs

The service VMs play a crucial role as target servers that are to be automatically configured. Building upon the experience gained from setting up the management VMs, the deployment of service VMs is also managed through Vagrant by crafting a Vagrantfile. Within the scope of this internship, we generate three distinct service VMs. These VMs play a dual role; not only do they serve as targets for security compliance scanning, but they are also integral components of the forthcoming PenTest Lab.

Since service VMs are controlled through SSH connections, it is imperative that they are connected via a shared virtual network alongside the VM running GitLab Runner. Additionally, to facilitate secure access, a public SSH key must also be added as an authorized key upon the creation of each service VM. Furthermore, all three VMs are provisioned with Debian 11, aligning with the prevailing operating system adopted by most VMs within ESEO's cloud infrastructure. With these requirements in place, we can efficiently design and execute the Vagrantfile to set up the service VMs. This step finalizes the setup stage and results in a local environment as illustrated in Figure 3.2.

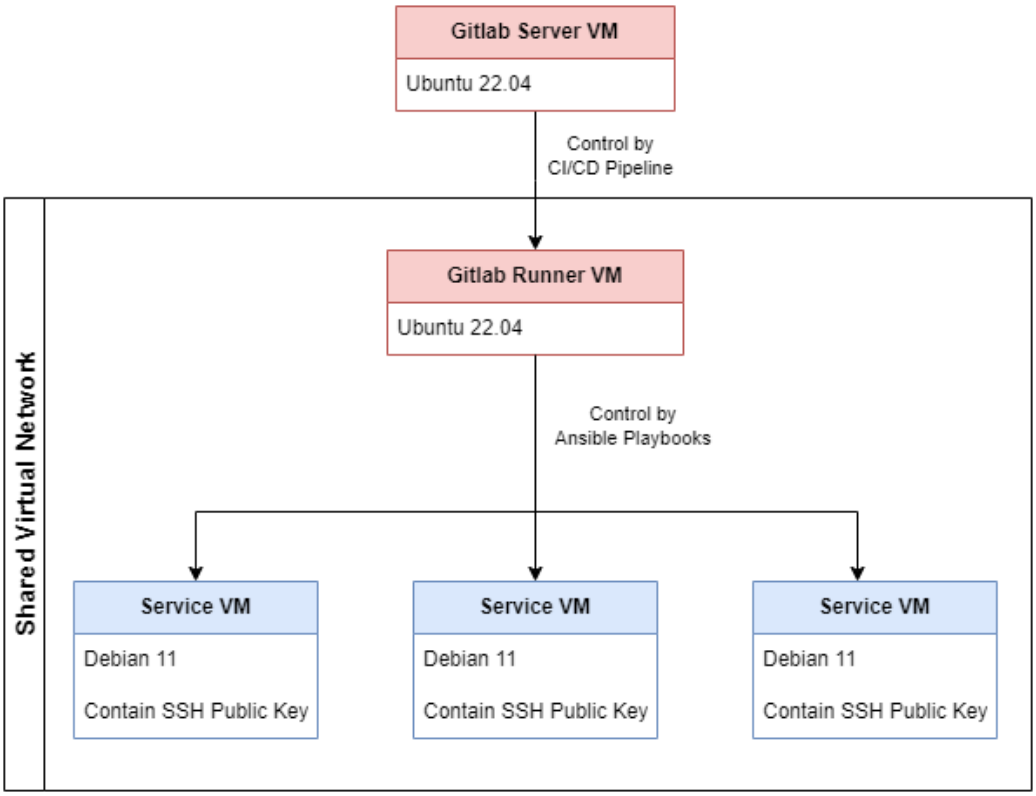

Figure 3.2 – Local environment mirroring ESEO infrastructure

3.2 Methodology

With the environment setup completed, the next phase involves the creation of the repositories outlined in our project objectives. All three repositories require the development of GitLab CI/CD pipelines, achieved by making a file named `.gitlab-ci.yml` in the root directory. This file serves as a list defining the series of CI/CD jobs to be executed. Examples of these jobs include building tools or executing Ansible Playbooks to control remote servers. For this specific project, most CI/CD jobs center on the execution of playbooks to configure the service VMs. Consequently, our workflow centers around a continuous loop: developing pipelines and playbooks, committing changes to the GitLab repository, and executing jobs to assess and identify necessary adjustments.

In order to execute jobs and make use of these repositories, we need to provide the required CI/CD variables through the project settings section. These variables include essential components such as Ansible configuration and inventory files, ensuring the project can be customized to suit any infrastructure. In addition, sensitive components such as the SSH private key can also be securely stored as variables instead of being embedded within the source code. Following the setup of these variables, the jobs can be initiated by submitting them to the GitLab Runner via the Server's web interface. The Runner is the underlying machine responsible for executing CI/CD jobs. It performs these tasks within temporary Docker containers and, upon completion, returns the results to the GitLab Server.

3.3 OpenSCAP Integration

As previously discussed, OpenSCAP is a scanning tool that assesses the security configuration settings of a system. It accomplishes this by examining the system against predefined rules, which are established in accordance with specific standards or security policies.

To integrate OpenSCAP and conduct security scans for the service VMs, we've designed a pipeline comprising three main jobs: building OpenSCAP, constructing the SCAP Security Guide (SSG), and executing scans on target machines. The flexibility of this pipeline allows us to initiate these jobs manually based on our specific requirements. This means we can run scans as needed while only updating the tools when newer versions are available. Moreover, after building the tools, we package and store them on the GitLab Server, making tool installation on different service VMs more efficient by avoiding repeatedly building the same version.

The first job, which involves building OpenSCAP, does not necessitate access to the service VMs. Alternatively, it is scripted and executed locally by the GitLab Runner within a Docker container, utilizing the Debian 11 image. In this process, we make a request to GitHub API to query the tag name of the latest OpenSCAP release. This tag name is then compared with the version we previously built and stored on our GitLab Server. The script exits if the version tags match, indicating no update is needed. However, if it's our initial execution or a newer version is available, the script proceeds with the OpenSCAP build process. This entails installing all essential dependencies, fetching the latest source code from the official GitHub repository, and compiling the tool. The resulting binary is then built into a Debian package. A test is subsequently conducted by employing this package to install OpenSCAP within the container. If the installation is successful, the package is pushed to a dedicated repository on our GitLab Server, tagged with the latest version.

The procedure for building SSG closely mirrors that of OpenSCAP. We applied the same method of checking versions for both SSG and OpenSCAP. This is because constructing SSG necessitates the presence of OpenSCAP along with other essential dependencies. Subsequently, the script downloads SSG from its GitHub repository and builds it using a Makefile. Building from source code offers the added advantage of time efficiency, enabling us to generate security guides tailored to our specific operating system rather than making guides for all supported OS. The resulting files that contain security policies are relatively compacted and not in binary form, so they can be directly stored in a GitLab repository, omitting the need for packaging. However, prior to pushing these security policies to our GitLab Server, we execute a test scan inside the container to verify that our tools are fully functional and ready for use.

With the tools prepared, we can initiate the compliance scan on the service VMs. This process also starts through a script executed by the GitLab Runner. However, because this process involves remotely managing VMs using Ansible, we must use a specific Docker image,

"`theohbrothers/docker-ansible:v2.10.7-alpine-3.13`," instead of the Debian 11 image. This is the exact image utilized by ESEO and is based on Alpine Linux OS, already pre-installed with Ansible.

The script's initial step involves verifying the versions of the tools, proceeding only if they are confirmed to be up-to-date. Subsequently, it downloads the Debian packages of OpenSCAP and the security policies into the Docker container, preparing them for transfer to the service VMs. Finally, it executes a security scan across the target machines specified in the inventory file by running an Ansible playbook.

The tasks within the playbook are executed in parallel on each service VM. Initially, we check whether OpenSCAP is either not installed or requires an update. In such cases, the Debian package is copied to the VM to reinstall OpenSCAP before proceeding with the scan. When OpenSCAP is confirmed to be up-to-date, the playbook transfers our selected security policies to target VMs. For this project, we aimed to evaluate the system against the configuration recommendations from ANSSI (the French National Information Systems Security Agency), which corresponds to a profile named `content_profile_anssi_np_nt28_average`. With all requisite components in place, the playbook initiates the scan and retrieves the reports back to the GitLab Runner. These reports are organized based on the service VM hostname and are saved as artifacts on the GitLab Server for further analysis.

3.4 PenTest Lab Deployment

In this subsequent phase of the project, our focus shifts to the deployment of the PenTest Lab by configuring two service VMs. The first VM, named "vpnserver," will be equipped with OpenVPN to control access to the lab. The second VM, denoted as "backend," will be configured with Docker to serve as the hosting environment for all the vulnerable containers that constitute the lab. In addition to housing these vulnerable containers, the backend VM will also include a container named "attacker." To access and utilize the lab, students must follow a two-step process: first, they connect to the VPN server, and then, they establish a connection to the "attacker" container via SSH. As implied by its name, the "attacker" container is equipped with various tools that enable students to conduct penetration testing on other vulnerable containers within the lab environment.

The deployment of the PenTest Lab is an automated process encompassing four distinct CI jobs. Each job corresponds to the execution of a specific Ansible playbook:

- `create_certificates`: This playbook generates all the necessary cryptographic keys and certificates required for the VPN server.

- `config_vpnserver`: This playbook configures the VPN server and establishes firewall rules on the "vpnserver" VM, enabling secure remote access.
- `create_clients`: This playbook is responsible for creating client configuration files for all users who will access the VPN.
- `config_backend`: This playbook sets up the "backend" server. It installs Docker and initiates the necessary containers that make up the lab environment.

3.4.1 Certificates Generation

OpenVPN operates using a Public Key Infrastructure (PKI) an authentication technology that relies on trusted parties to digitally sign documents certifying the association of cryptographic keys with users or devices. The key can then be used as an identity for the user in digital networks. The trusted entity in this context is known as a Certificate Authority (CA). The CA possesses its own cryptographic key for signing documents that link public keys to devices, known as certificates [14].

The playbook `create_certificates` starts by generating two key pairs. The first key pair corresponds to a master certificate and key, which belongs to a CA and serves to sign other certificates. The second key pair belongs to the VPN server, enabling it to be identified before establishing connections with clients. These certificates and keys are created using easy-rsa, a tool recommended by OpenVPN for building and managing PKI, and they are securely stored on the VPN server. A more secure practice is to create the keys on a separate server acting as a Certificate Authority, distinct from the VPN server. This CA server can even be disconnected from the Internet for heightened security. Once created, all essential keys are transferred to the VPN server. Meanwhile, the most sensitive key, the private CA key used for signing all other certificates, is retained exclusively on the CA server since the VPN server only requires the public key for signature verification. While our playbook could easily be adapted to follow this practice, we opted to generate the keys directly on the VPN server to avoid the complexity of setting up an additional Virtual Machine. Furthermore, the automation pipeline allows us to enhance security by periodically regenerating the keys as an alternative approach.

In OpenVPN, the RSA certificates and keys are only employed for authentication purposes. Communication data, on the other hand, is encrypted using the Diffie-Hellman (DH) key exchange method. Consequently, easy-rsa is again used to generate DH parameters, which are stored on the VPN server as a file called "dh.pem." After the authentication between the server and clients, "dh.pem" is transmitted to clients, enabling them to generate a shared pre-master secret. Subsequently, a master secret is derived from this pre-master secret to facilitate the encryption of communication data.

Lastly, the playbook generates a key for TLS authentication, a key that is stored by both the VPN server and clients. This key adds an additional HMAC signature to all SSL/TLS handshake packets, serving as a mechanism for integrity verification. Any UDP packet not bearing the correct HMAC signature can be dropped without further processing, saving time and computing resources.

3.4.2 VPN Server Configuration

There are five keys generated from the previous CI job that are essential for configuring the VPN server: the public CA key, the VPN server's private and public keys, the DH parameters, and the TLS authentication key. These keys can be either transferred from another machine or, as in our case, generated directly on the VPN server and then copied to a designated configuration directory. Once these keys are prepared, the configuration of the VPN can be automated using the playbook `config_vpnserver`. This process begins with installing two critical tools: OpenVPN and UFW.

The playbook then proceeds to configure OpenVPN by generating a configuration file based on a template that incorporates various variables. This template is derived from the sample server config file provided by OpenVPN, with modifications to include these following relevant variables.

```
---
vpn_config_dir: "/etc/openvpn/server"
ca_crt_path: "{{ vpn_config_dir }}/ca.crt"
server_crt_path: "{{ vpn_config_dir }}/server.crt"
server_key_path: "{{ vpn_config_dir }}/server.key"
dh_pem_path: "{{ vpn_config_dir }}/dh.pem"
ta_key_path: "{{ vpn_config_dir }}/ta.key"

private_interface: "eth1"
private_network: "10.0.0.0"
private_netmask: "255.255.255.0"
private_netcidr: "24"

vpn_network: "10.8.0.0"
vpn_netmask: "255.255.255.0"
vpn_netcidr: "24"
```

The first set of variables specifies the paths to the required keys, while the remaining variables define network configurations. Variables prefixed with "private" relate to the network linking the VPN server and the backend server, while those prefixed with "vpn" refer to the virtual network connecting clients and the VPN server. Additionally, an essential modification to the template involves adding the following line:

```
push "route {{ private_network }} {{ private_netmask }}"
```

This configuration instructs the VPN server to push a route to the private network, enabling clients to reach the backend server once connected to the VPN.

Following the OpenVPN configuration, the playbook establishes firewall rules that adhere to a whitelist policy. Under this policy, all incoming traffic is blocked except for specific deliberately allowed traffic. Initially, since the VPN server runs SSH and OpenVPN, the firewall permits traffic to the respective ports of these services. In addition, as shown in Figure 3.3, student traffic to the backend server must pass through the VPN server. Therefore, the VPN server is configured to enable IPv4 forwarding. Moreover, as previously mentioned, students can solely access the backend server via an SSH connection to the "attacker" container, which listens on port 22. Since the container's port is mapped to port 2222 on the backend server, the firewall authorizes clients to connect to this server on port 2222 exclusively. Subsequently, the VPN server is configured to masquerade packets originating from clients before forwarding them to the backend server via the private network. In other words, the VPN server alters the source IP address of packets to its own address before forwarding, ensuring that responses from the backend server are also routed through itself.

With all rules and settings in place, the final step involves enabling UFW and starting the OpenVPN service, thus completing the configuration of the VPN server.

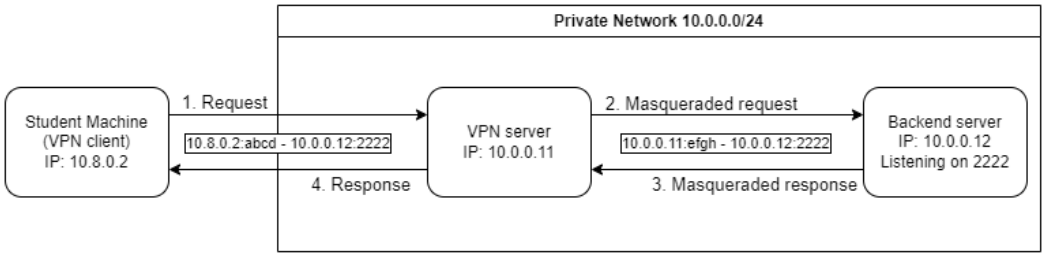

Figure 3.3 – Packets from VPN clients being masqueraded before reaching backend server

3.4.3 Clients Generation

To generate multiple client configuration files for students, we employed the `create_clients` playbook. It's important to note that this playbook relies on the existence of the master key pairs and should be executed on the same machine used for generating those keys. By utilizing this playbook, we can efficiently create configuration files for many clients simultaneously by specifying a list of clients as follows:

```
vpn_clients:
- "client1"
- "client2"
```

With the list above, two configuration files, "client1.ovpn" and "client2.ovpn," will be automatically generated based on a template file and stored as artifacts on the GitLab Server. The playbook creates a VPN certificate and private key for each client and embeds these credentials in their respective configuration files. In addition, the CA certificate, along with the TLS authentication key, is also included in each of those files. Upon completing this step, the resulting "ovpn" files are ready for secure distribution to students, providing them access to the VPN server.

3.4.4 Backend Server Configuration

The `config_backend` playbook serves two primary objectives: installing Docker and initiating prepared containers. The playbook starts by installing Docker, the core engine for containerization. The Ansible tasks for installing the main Docker engine and the Docker Compose plugin follow the instructions provided in the official Docker documentation. Notably, these tasks are encapsulated within a role named "docker-installer," designed to be reusable, simplifying Docker installations in other projects. After Docker is successfully installed, the playbook proceeds to copy a folder named "pentest-lab" to the backend server.

Challenges in the lab are provided through various containers, each organized within a separate subfolder inside the "pentest-lab" directory. Each subfolder contains a docker-compose file and any other necessary configuration files. These docker-compose files are included in a primary docker-compose located at the root of the "pentest-lab" directory. This folder structure simplifies adding or removing challenges to solely editing the primary docker-compose file. It is important to note that if the "pentest-lab" folder ever requires modifications, this playbook can be reused seamlessly. The playbook includes a check to determine whether Docker is already installed on the backend server, thus enabling it to skip installation steps before automatically rebuilding or removing containers corresponding to the changes.

Another critical aspect of deploying the backend server is to select challenges for the lab. ESEO's interests are divided into two types of challenges: real-world challenges, requiring students to research and exploit actual Common Vulnerabilities and Exposures (CVEs), and common challenges designed for students to practice their acquired knowledge.

For the first type of challenge, we opted to source them from Vulhub, a GitHub repository serving as a library of configuration files to set up different Docker containers. These containers are designed to mirror particular real-world CVEs. Within the scope of our internship, we selected nine containers, each associated with CVEs from technologies relevant to ESEO's teaching activities, namely Bash, Phpmyadmin, PHP, and Tomcat. We adapted the provided configuration files and included step-by-step exploit instructions for each challenge before organizing them into folders based on their underlying technologies.

For the second type of challenge, we leveraged reputable and well-tested open-source projects, including WebGoat, JuiceShop, and bWAPP. WebGoat is a deliberately insecure application maintained by the Open Worldwide Application Security Project (OWASP). WebGoat includes instructional lessons addressing nearly all OWASP Top 10 vulnerabilities as well as vulnerabilities commonly encountered in Java-based applications. Meanwhile, JuiceShop is written exclusively in JavaScript and claims to be “the most modern and sophisticated insecure web application.” It encompasses vulnerabilities from the entire OWASP Top Ten, along with many other security flaws found in real-world applications. Lastly, by relying on the LAMP (Linux, Apache, MySQL, PHP) stack, bWAPP offers over 100 web vulnerabilities, each with adjustable security levels. These three projects already have their official docker image built, so their integration is simplified by creating docker-compose files that utilize these images. Since these projects are web-based, but students can only access the lab through a command line interface, instructions to forward ports to students’ computers are also provided in each project’s respective folder.

The last component of the lab, which is indispensable, is the "attacker" container. A Dockerfile is crafted to build the image for this container, enabling control over the installed tools. The image is based on the latest Debian image and comes pre-installed with standard software, including Git, Curl, Vim, Python3, Golang, VScode server, Nmap, etc. To facilitate SSH connections when initiated, the image also incorporates the OpenSSH server with password authentication enabled. The Dockerfile and docker-compose files for the "attacker" container are also prepared within the "pentest-lab" directory, allowing its container to be started concurrently with the lab’s challenges.

3.5 Wazuh Monitoring Setup

Wazuh offers a detailed and straightforward method of deploying this monitoring system with Ansible. Therefore, as a final touch to our internship, we decided to adapt the Wazuh deployment playbooks into a CI/CD pipeline. The motivation for this decision was underpinned by the observation that this integration does not demand much effort but holds the potential to significantly enhance the security of several systems. Our CI/CD pipeline consists of two jobs: deploying Wazuh central components and deploying Wazuh agents, both of which utilize the Docker container “theohbrothers/docker-ansible:v2.10.7-alpine-3.13” to execute Ansible playbooks. To develop and test our pipeline, we aimed to monitor our newly configured PenTest Lab. This was accomplished by deploying Wazuh agents on both the backend and vpnserver virtual machines (VMs). Simultaneously, the central components of Wazuh were installed on the remaining service VM, denoted as "wazuhserver." These tasks were effectively automated by employing two playbooks, wherein we simply specified the roles provided by Wazuh. While the playbooks themselves were readily available and documented on GitHub, the challenge lies in integrating them into our repository. Our chosen approach was to incorporate the stable branch of the Wazuh repository as a submodule. Submodule is a Git utility that facilitates the referencing of another repository as a subdirectory.

This approach allows us to reduce the size and complexity of our repository while benefiting from the latest updates and patches of the official Wazuh repository. It's noteworthy that, to clone our repository with the submodule included, the "–recurse-submodules" tag must be employed. Additionally, we encountered complexities in customizing the playbooks to execute within an Alpine Linux Docker container, deviating from their initial compatibility with a Debian-based environment. This challenge surfaced particularly when tasks were delegated to run locally inside the container rather than on the remote machines. To address this problem, we had to identify and install the missing essential tools, such as Bash, Openssl, Git, etc., in Alpine Linux prior to executing the playbooks.

Section 4

Results

4.1 Git Repositories

As illustrated in Figure 4.1, we successfully completed our objectives and developed three repositories on our GitLab server to perform the automated tasks. Additionally, these repositories were also assembled and published on GitHub at https://github.com/mindt102/bachelor-internship, as shown in Figure 4.2, to share the results and contribution of our work with the IT community.

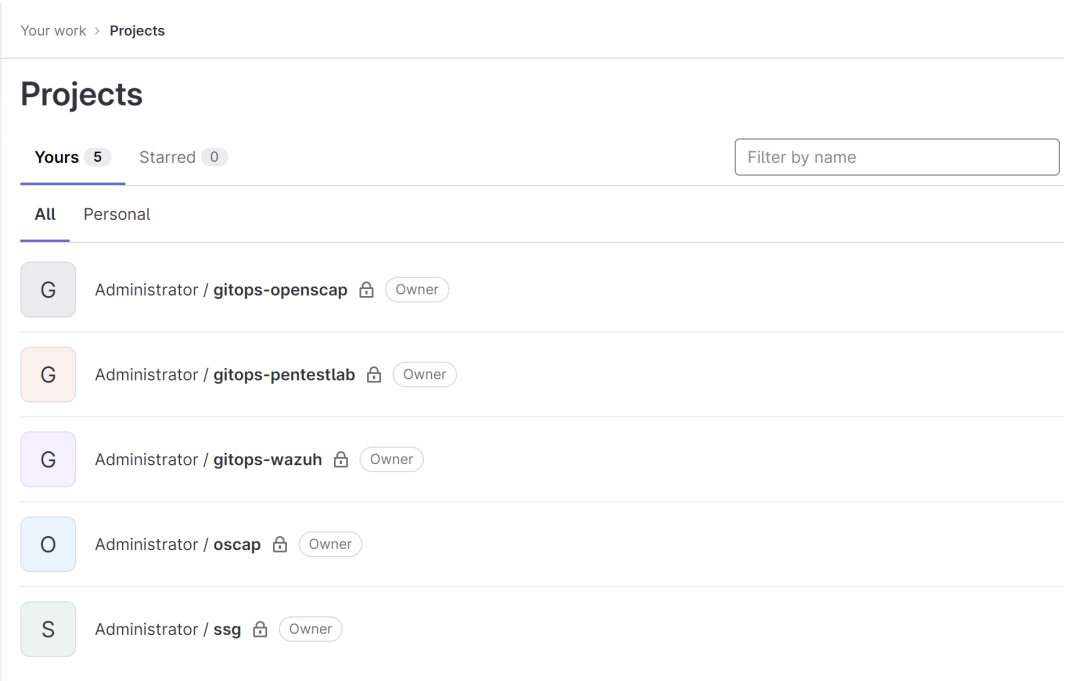

Figure 4.1 – Git Repositories on GitLab Server

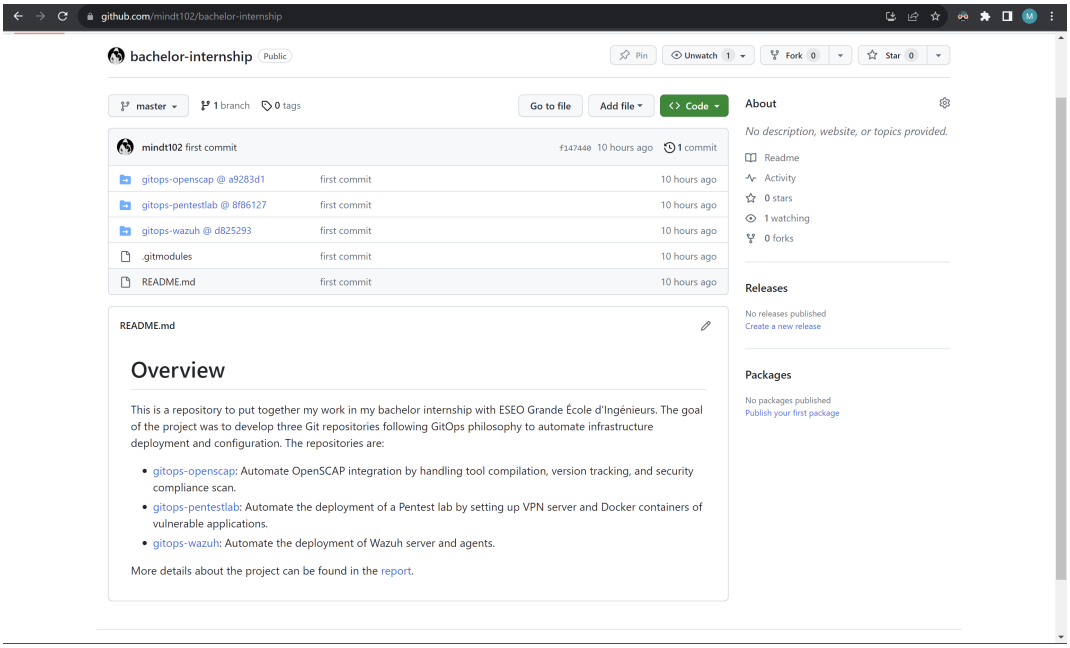

Figure 4.2 – Git Repositories on GitHub

4.2 OpenSCAP Compliance Scan

The three-stage pipeline to integrate OpenSCAP is visually represented in Figure 4.3.

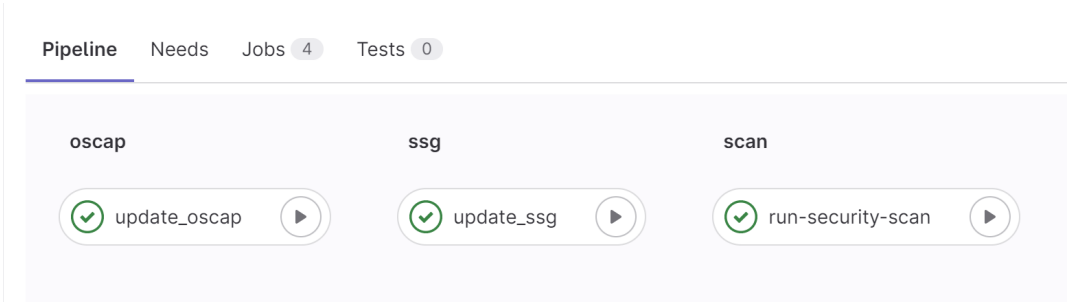

Figure 4.3 – OpenSCAP Integration Pipeline

Using the initial two CI jobs, we managed to build and store different versions of the required tools, as can be seen in Figure 4.4 and 4.5.

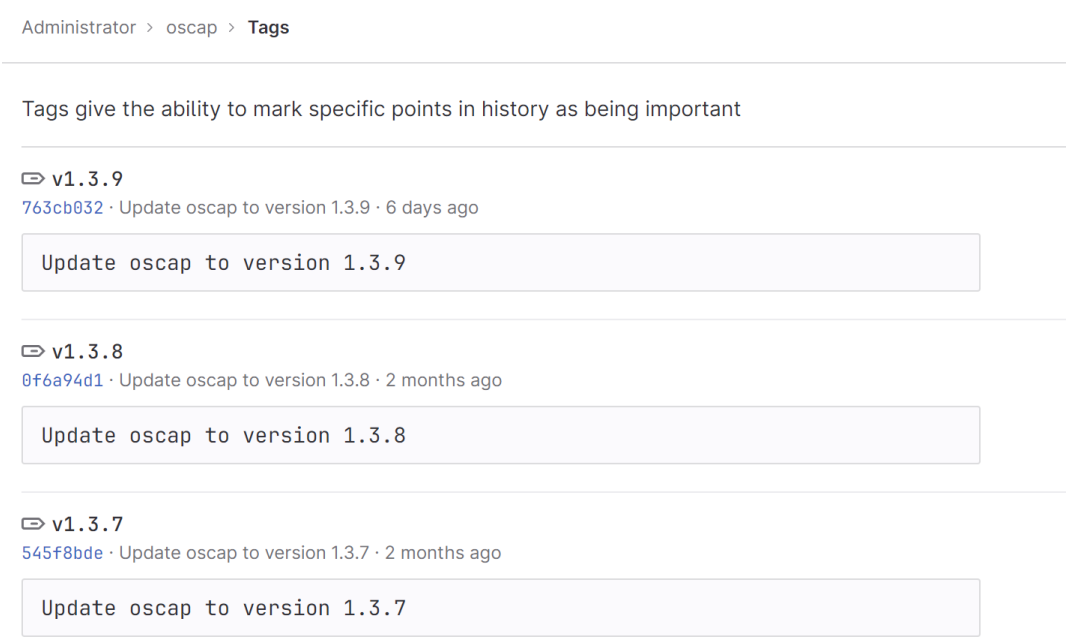

Figure 4.4 – OpenSCAP versions automatically updated using CI jobs

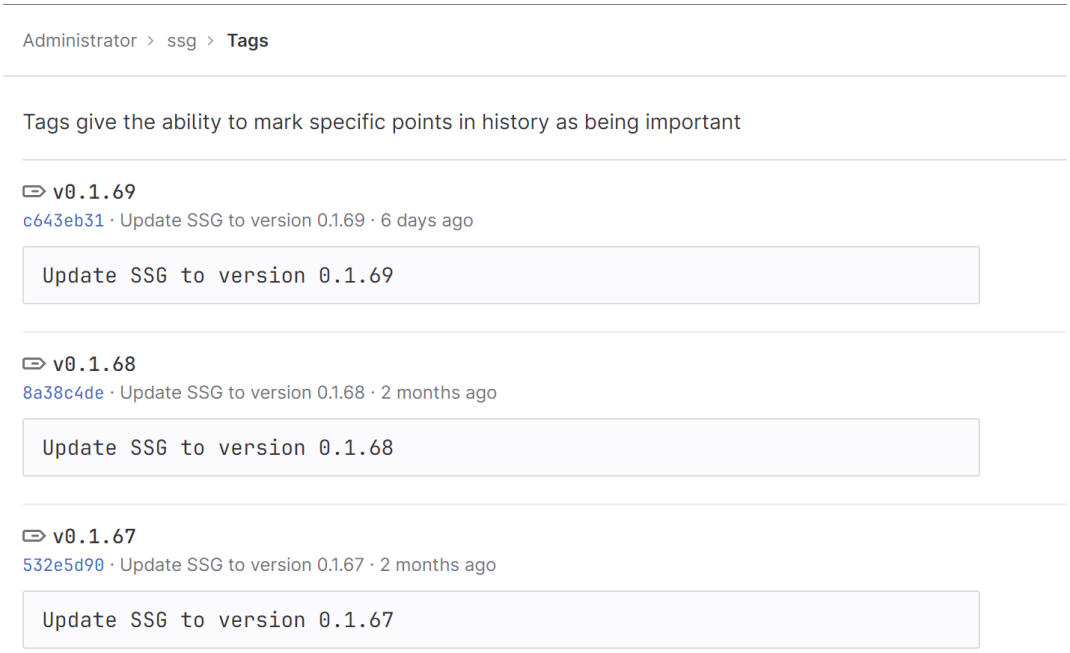

Figure 4.5 – SCAP security guide versions automatically updated using CI jobs

With the third job, we scanned all three service VMs and collected the reports on the GitLab server. Figure 4.6 displays a representative sample of this report. These scans provide insight into the number of passed rules, the severity of the failed rules, as well as recommend scripts for remediation.

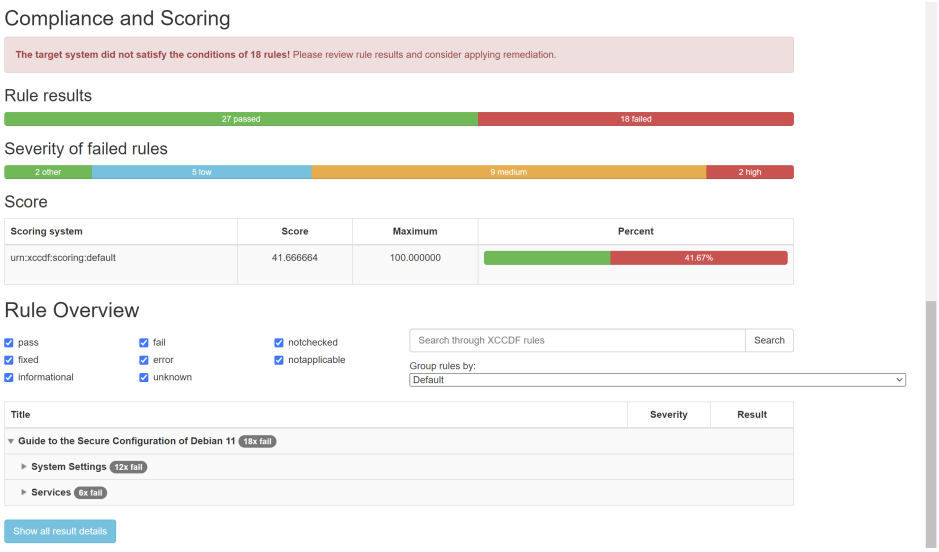

Figure 4.6 – Example OpenSCAP compliance scan report

4.3 PenTest Lab Usage

Our four CI jobs designed to execute Ansible playbooks are depicted in Figure 4.7.

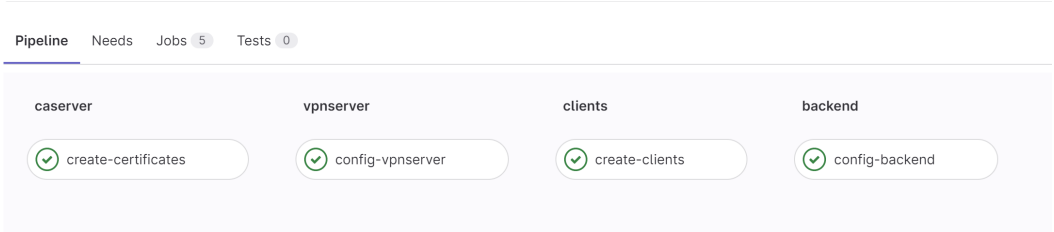

Figure 4.7 – PenTest Lab Deployment Pipeline

The VPN server configuration, complete with firewall rules, was executed successfully, allowing students to establish secure SSH connections to the "attacker" container. This connection is evidenced in Figure 4.8 and 4.9.

```
PS C:\Users\Admin\Downloads> openvpn.exe .\client1.ovpn
2023-09-19 00:19:33 DEPRECATED OPTION: --cipher set to 'AES-256-CBC' but missing in --data-ciphers (AES-256-GCM:AES-128-GCM:CHACHA20-POLY1305). OpenVP
N ignores --cipher for cipher negotiations.
2023-09-19 00:19:33 OpenVPN 2.6.3 [git:v2.6.3/94aad8c51043a805] Windows-MSVC [SSL (OpenSSL)] [LZO] [LZ4] [PKCS11] [AEAD] [DCO] built on Apr 13 2023
2023-09-19 00:19:33 Windows version 10.0 (Windows 10 or greater), amd64 executable
2023-09-19 00:19:33 library versions: OpenSSL 3.1.0 14 Mar 2023, LZO 2.10
2023-09-19 00:19:33 DCO version: v0
2023-09-19 00:19:33 TCP/UDP: Preserving recently used remote address: [AF_INET]192.168.11.11:1194
2023-09-19 00:19:33 ovpn-dco device [OpenVPN Data Channel Offload] opened
2023-09-19 00:19:33 UDP link local: (not bound)
2023-09-19 00:19:33 UDP link remote: [AF_INET]192.168.11.11:1194
2023-09-19 00:19:33 TLS: Initial packet from [AF_INET]192.168.11.11:1194, sid=3eb6546e 4001a9b2
2023-09-19 00:19:33 VERIFY OK: depth=1, CN=Easy-RSA CA
2023-09-19 00:19:33 VERIFY KU OK
2023-09-19 00:19:33 Validating certificate extended key usage
2023-09-19 00:19:33 ++ Certificate has EKU (str) TLS Web Server Authentication, expects TLS Web Server Authentication
2023-09-19 00:19:33 VERIFY EKU OK
2023-09-19 00:19:33 VERIFY OK: depth=0, CN=server
2023-09-19 00:19:33 Control Channel: TLSv1.3, cipher TLSv1.3 TLS_AES_256_GCM_SHA384, peer certificate: 2048 bit RSA, signature: RSA-SHA256
2023-09-19 00:19:33 [server] Peer Connection Initiated with [AF_INET]192.168.11.11:1194
2023-09-19 00:19:33 TLS: move_session: dest=TM_ACTIVE src=TM_INITIAL reinit_src=1
2023-09-19 00:19:33 TLS: tls_multi_process: initial untrusted session promoted to trusted
2023-09-19 00:19:33 PUSH: Received control message: 'PUSH_REPLY,route 10.0.0.0 255.255.255.0,route 10.8.0.1,topology net30,ping 10,ping-restart 120,if
config 10.8.0.6 10.8.0.5,peer-id 0,cipher AES-256-GCM'
2023-09-19 00:19:33 OPTIONS IMPORT: --ifconfig/up options modified
2023-09-19 00:19:33 OPTIONS IMPORT: route options modified
2023-09-19 00:19:33 interactive service msg_channel=0
2023-09-19 00:19:33 NETSH: C:\WINDOWS\system32\netsh.exe interface ip set address 17 static 10.8.0.6 255.255.255.252
2023-09-19 00:19:33 NETSH: C:\WINDOWS\system32\netsh.exe interface ip delete dns 17 all
2023-09-19 00:19:33 NETSH: C:\WINDOWS\system32\netsh.exe interface ip delete wins 17 all
2023-09-19 00:19:33 IPv4 MTU set to 1500 on interface 17 using SetIpInterfaceEntry()
2023-09-19 00:19:33 C:\WINDOWS\system32\route.exe ADD 10.0.0.0 MASK 255.255.255.0 10.8.0.5 METRIC 200
2023-09-19 00:19:33 Route addition via ipapi [adaptive] succeeded
2023-09-19 00:19:33 C:\WINDOWS\system32\route.exe ADD 10.8.0.1 MASK 255.255.255.255 10.8.0.5 METRIC 200
2023-09-19 00:19:33 Route addition via ipapi [adaptive] succeeded
2023-09-19 00:19:33 Initialization Sequence Completed
2023-09-19 00:19:33 Data Channel: cipher 'AES-256-GCM', peer-id: 0
2023-09-19 00:19:33 Timers: ping 10, ping-restart 120
```

Figure 4.8 – Example connection to VPN server

```
PS C:\Users\Admin> ssh student@10.0.0.12 -p 2222
student@10.0.0.12's password:
Linux 11b99348b273 5.10.0-22-amd64 #1 SMP Debian 5.10.178-3 (2023-04-22) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Sep 18 17:23:45 2023 from 10.0.0.11
student@11b99348b273:~$
```

Figure 4.9 – Example SSH connection to the "attacker" container as student

Figure 4.10 provides a demonstration of how students can exploit the Shellshock vulnerability from a Docker container of the same name, effectively executing the command "id" remotely. Shellshock, a vulnerability dating back to 2014, underscores a security risk associated with older versions of Bash, rendering them susceptible to remote command execution. This vulnerability has been reproduced as one of many challenges hosted on our backend VM.

```
student@11b99348b273:~$ curl http://shellshock/victim.cgi
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Bash ShellShock</title>
</head>
<body>
<p>
Hello world
</p>
</body>
</html>
student@11b99348b273:~$ curl http://shellshock/victim.cgi -A "() { foo; }; echo Content-Type: text/plain; echo; /usr/bin/id"
uid=33(www-data) gid=33(www-data) groups=33(www-data)
student@11b99348b273:~$
```

Figure 4.10 – Example exploit of shellshock vulnerability

Figure 4.11 showcases the accessibility of the WebGoat web interface to students from their local machines, enabling students to review and practice vulnerabilities featured in this open-source

project. It is important to highlight that students must initiate port forwarding to their local machines before gaining access to the web interface using the following command:

```
ssh -L 8080:webgoat:8080 -L 9090:webgoat:9090 student@10.0.0.12 -p 2222
```

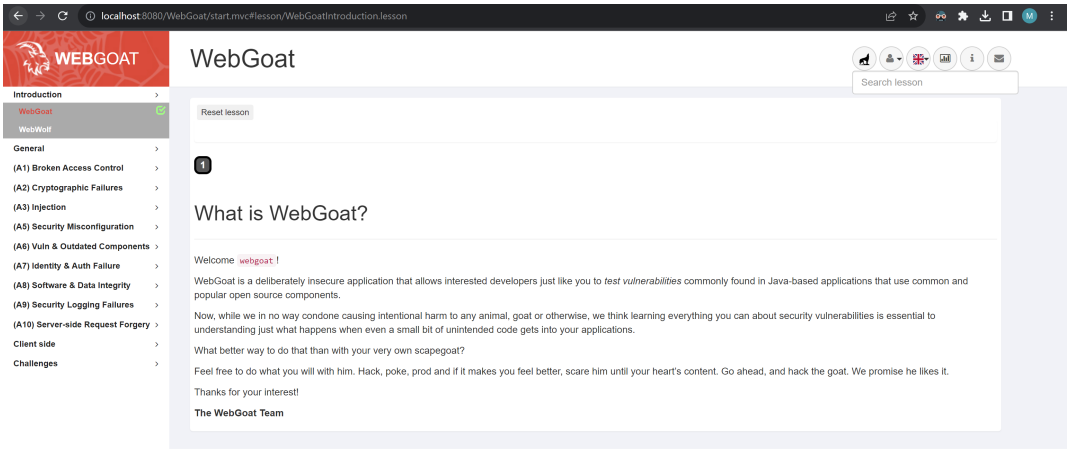

Figure 4.11 – WebGoat web interface

4.4 Wazuh Monitoring

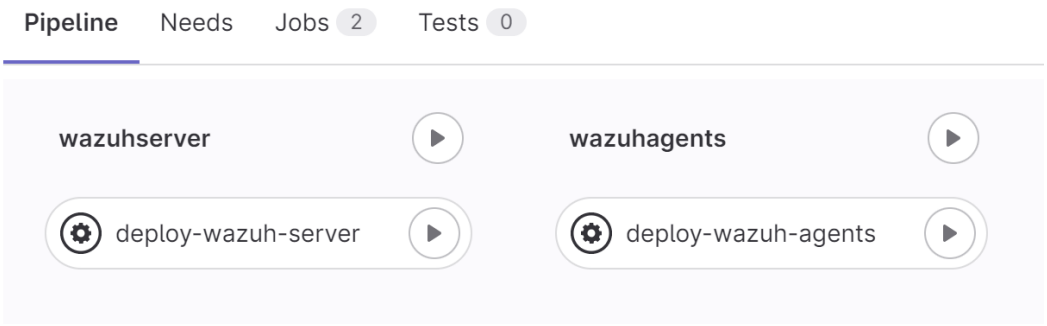

Figure 4.12 – Wazuh Deployment Pipeline

Figure 4.13 presents the Wazuh Dashboard, one of the central components that was successfully deployed on our designated "wazuhserver" VM. This dashboard serves as a central hub for monitoring and managing security events. Notably, this figure displays the presence of two actively monitored agents. These active agents signify that both the "vpnserver" and "backend" VMs are under continuous monitoring by the Wazuh server, thus proving the effectiveness of our Wazuh deployment pipeline (Figure 4.12).

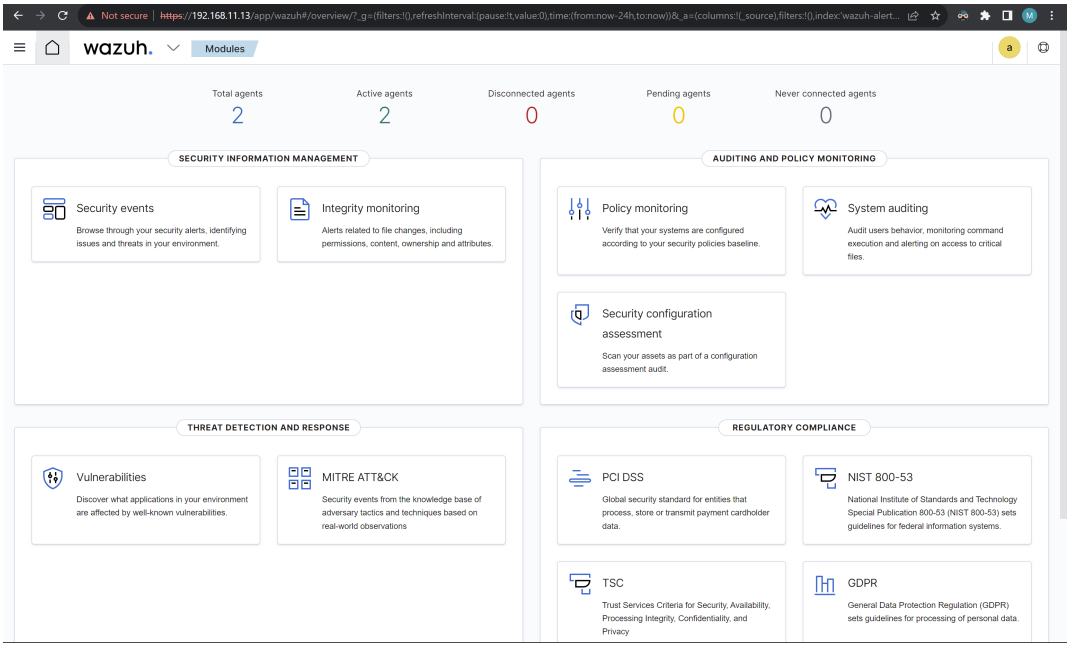

Figure 4.13 – Wazuh Dashboard

Section 5

Discussion

5.1 Conclusion

In conclusion, by integrating OpenSCAP, deploying the PenTest Lab, and adopting the Wazuh monitoring system, this project not only addresses the security challenges faced by ESEO but also provides an efficient method to deploy a controlled environment for cybersecurity education.

This internship project also demonstrates the power of GitOps in automating infrastructure management tasks. The outcomes of this project serve as an example of the benefits of applying modern DevOps practices to the field of cybersecurity and infrastructure management, making them accessible to a broader audience.

5.2 Discussion

By creating three distinct Git repositories, each aligned with GitOps principles, we've not only fulfilled the project's objectives but have also opened up the possibility for other individuals and organizations to benefit from our work. These repositories, hosted publicly on GitHub, serve as valuable resources for efficiently deploying similar infrastructures and addressing security challenges.

As explained above, our project employs the same GitLab server to store the OpenSCAP and SCAP security guide repositories post-compilation. To replicate this setup, two repositories, namely "oscap" and "ssg" must be established on your GitLab server. Subsequently, access tokens must be generated for these repositories and securely stored as GitLab secret variables, referred to as `OSCAP_ACCESS_TOKEN` and `SSG_ACCESS_TOKEN`, respectively. Additionally, it might be required to replace our illustrative URL, "gitlab.example.com," with the actual URL of your GitLab server. In case a different approach is employed for storing these tools, the steps for storing and retrieving them in oscap.yml, ssg.yml, and .gitlab-ci.yml need to be adapted accordingly.

Our PenTest Lab deployment method can be promptly implemented by cloning the repository from GitHub and configuring the IP addresses for your VPN and backend server. Step-by-step guides for exploiting CVE-based challenges are included within each target's dedicated folder. The selection of currently available targets, as depicted in the Table 5.1, is rooted in their alignment

with the curriculum of our host institution. Nevertheless, docker-compose files can be effortlessly incorporated or removed from the “pentest lab” folder to tailor the target containers to your specific requirements.

Table 5.1 – Available CVE-based challenges

Vulnerable Software	Target Container	Description
phpmyadmin	cve-2016-5734	PhpMyAdmin 4.0.x—4.6.2 Remote Code Execution Vulnerability (CVE-2016-5734)
	cve-2018-12613	phpmyadmin 4.8.1 Remote File Inclusion Vulnerability (CVE-2018-12613)
	wooyun-2016-199433	Phpmyadmin Scripts/setup.php Deserialization Vulnerability (WooYun-2016-199433)
bash	shellshock	Shellshock Remote Command Injection (CVE-2014-6271)
tomcat	cve-2017-12615	Tomcat Arbitrary Write-file Vulnerability through PUT Method (CVE-2017-12615)
	backend-rce	Tomcat7+ Weak Password && Backend Getshell Vulnerability
php	8.1-backdoor	PHP 8.1.0-dev User-Agentt Backdoor
	cve-2019-11043	PHP-FPM Remote Command Execution (CVE-2019-11043)
	inclusion	PHP Local File Inclusion RCE with PHPINFO

To utilize the Wazuh deployment repository, it is crucial to recursively clone the submodules to retrieve the latest playbooks from Wazuh. This repository also can be used immediately after specifying the address of Wazuh agents and the Wazuh server. All our central components are deployed on a single machine, which, according to Wazuh documentation, is suitable for monitoring up to 100 endpoints and retaining alert data for up to 90 days. Consulting the Wazuh documentation and adjusting to the Wazuh server deployment playbooks is advised for scenarios necessitating a distributed deployment to oversee larger systems.

5.3 Future Works

While the current project has successfully achieved its primary objectives, there is room for future work that can further enhance the functionality and educational value of the system.

Firstly, a promising area for future development involves automating the execution of remediation scripts based on recommendations derived from OpenSCAP scans. Currently, the integration of OpenSCAP provides security audits, but automating the execution of remediation scripts can help administrators proactively address identified vulnerabilities, reducing the response time. This automation would reduce the manual effort required for system hardening and enhance the overall security of the cloud environment.

Secondly, to improve the user experience within the PenTest Lab, future work should consider modifying the "attacker" container to support a Graphical User Interface (GUI). This enhancement will equip students with a more intuitive and interactive environment for practicing penetration testing techniques. A GUI can offer visual tools and interfaces that make the lab more accessible for students of varying technical backgrounds.

Lastly, in the realm of security monitoring, further research and optimization efforts are needed to tailor the Wazuh configuration specifically to the requirements of monitoring the PenTest Lab. By customizing the Wazuh setup to align with the unique challenges and activities within the lab, it's possible to enhance the precision and efficiency of threat detection and response mechanisms. This research can include the development of custom rules, alerts, and dashboards that provide administrators with real-time insights into security events and vulnerabilities within the PenTest Lab.

References

[1] *Ansible concepts*. URL: https://docs.ansible.com/ansible/latest/getting_started/basic_concepts.html.

[2] *Ansible Documentation*. URL: https://docs.ansible.com/ansible/latest/index.html.

[3] William Chia. *What is GitOps?* URL: https://about.gitlab.com/topics/gitops/.

[4] *Get started with GitLab CI/CD*. URL: https://docs.gitlab.com/ee/ci/index.html.

[5] *Is GitOps the next big thing in DevOps?* URL: https://www.atlassian.com/git/tutorials/gitops.

[6] Aswin Kumar KP. *Github vs Gitlab vs Bitbucket*. Dec. 2022. URL: https://disbug.io/en/blog/github-vs-gitlab-vs-bitbucket.

[7] *OpenSCAP Documentations Getting Started*. URL: https://www.open-scap.org/getting-started/.

[8] *OpenSCAP User Manual*. URL: https://static.open-scap.org/openscap-1.3/oscap_user_manual.html#_introduction.

[9] *SCAP Security Guide*. URL: https://www.open-scap.org/security-policies/scap-security-guide/.

[10] *Wazuh Components*. URL: https://documentation.wazuh.com/current/getting-started/components/index.html.

[11] *What is a container?* URL: https://docs.docker.com/get-started/#what-is-a-container.

[12] *What is a virtual machine (VM)?* URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-virtual-machine.

[13] *What is containerization?* URL: https://www.ibm.com/topics/containerization.

[14] *What is PKI (Public Key Infrastructure)?* URL: https://www.ssh.com/academy/pki.