jamk.fi

Automated Security Testing Utilizing Continuous Integration and Continuous Delivery Technologies

Pyry Koskela

Bachelor's thesis
May 2021
Information and communication technologies
Programme in Information and Communications Technology

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Koskela, Pyry	Bachelor's thesis	May 2021
		Language of publication: English
	Number of pages 55 + 8	Permission for web publication: Yes

Title of publication

Automated Security Testing Utilizing Continuous Integration and Continuous Delivery Technologies

Degree programme

Programme in Information and Communication Technology

Supervisor(s)

Saharinen, Karo; Nevala, Jarmo

Assigned by

Liveto Group Oy

Abstract

The circumstances of the year 2020 created a need for pushing everything and everyone online. To adapt to the situation at the required pace, the swift transition of existing systems and workflows from physical to digital was inevitable. However, the rapid development of new services resulted in the infrastructure not cutting the mustard in terms of information security and cybersecurity.

The objective was to develop and implement a security testing construct to secure the Client's expanding server and service infrastructure. The requirements specified for the security testing system were the ability to automate the testing implementation and adapt it to the Client's agile development workflow while making it as cost-efficient as possible.

The solution consisted of a containerized vulnerability assessment framework deployed into a dedicated server, designing and developing a CLI (Command-Line Interface) tool for remote interaction, and hooking the Client's CI/CD pipeline with the security testing service.

As an outcome of this constructive research approach, the Client's infrastructure includes a security testing service regularly scanning the servers against weaknesses with an ever-growing number of vulnerability tests.

The objective was achieved, and the solution provides a solid base for the Client's security testing. To further improve the reliability of the system, additional scanning tools could be implemented to run alongside.

Keywords/tags (subjects)

Cybersecurity, Testing, Automation, DevOps, DevSecOps, OpenVAS, GVM, Python, Docker, Container

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Koskela, Pyry	Opinnäytetyö, AMK	Toukokuu 2021
	Sivumäärä 55 + 8	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: Kyllä

Työn nimi

Automatisoitu tietoturvatestaus jatkuvan integraation ja jatkuvan toimituksen teknologioita hyödyntäen

Tutkinto-ohjelma

Tieto- ja viestintätekniikka

Työn ohjaaja(t)

Karo Saharinen, Jarmo Nevala

Toimeksiantaja(t)

Liveto Group Oy

Tiivistelmä

Vuoden 2020 olosuhteet loivat tarpeen siirtää kaiken Internettiin. Sopeutuaksemme tilanteeseen sen vaatimalla tahdilla, oli välttämätöntä muuttaa olemassaolevat järjestelmät ja työnkulut nopeasti fyysistestä digitaaliseen muotoon. Uusien palveluiden nopeatempoisen kehityksen seurauksena infrastruktuurit eivät kuitenkaan enää olleet vaadittavalla tasolla tieto- ja kyberturvan suhteen.

Tavoitteena oli kehittää ja toteuttaa tietoturvatestausrakenne Toimeksiantajan laajenevan palvelin- ja palveluinfrastruktuurin turvaamiseksi. Tietoturvatestausjärjestelmälle asetetut vaatimukset olivat testauksen automatisoitavuus, integraatio Toimeksiantajan ketterän kehitystyön menetelmiin sekä mahdollisuuksien mukainen kustannustehokkuus.

Ratkaisu koostui eriliselle palvelimelle asennetusta kontitetusta haavoittuvuusskannerista, skannerin etäkäyttöön tarkoitetun komentorivityökalun suunnittelusta ja kehittämisestä, sekä Toimeksiantajan jatkuvan integraation ja jatkuvan toimituksen (CI/CD) kanavan liittämisestä testauspalveluun.

Tämän konstruktiivisen tutkimuksen tuotoksena Toimeksiantajan infrastruktuuriin kuuluu tietoturvatestauspalvelu, joka skannaa palvelimien heikkouksia säännöllisesti jatkuvasti lisääntyvien haavoittuvuustestien avulla.

Tavoite saavutettiin ja ratkaisu tarjoaa vankan perustan Toimeksiantajan tietoturvatestaukselle. Järjestelmän luotettavuuden parantamiseksi voitaisiin rinnalle ottaa muita skannaustyökaluja.

Avainsanat (asiasanat)

Tietoturva, Testaus, Automaatio, DevOps, DevSecOps, OpenVAS, GVM, Python, Docker, Kontti

Muut tiedot (Salassa pidettävät liitteet)

Contents

Figures

Tables

Abbreviations

API	Application Programming Interface
CaaS	Containers as a Service
CD	Continuous Delivery, Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
FaaS	Function as a Service
GCP	Google Cloud Platform
GMP	Greenbone Management Protocol
GSA	Greenbone Security Assistant
GSE	Greenbone Source Edition
GUI	Graphical User Interface
GVM	Greenbone Vulnerability Management
GVMD	Greenbone Vulnerability Manager Daemon
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Transport Layer Security
IaaS	Infrastructure as a Service
IOCTA	Internet Organised Crime Threat Assessment
NVT	Network Vulnerability Test
OpenVAS	Open Vulnerability Assessment Scanner
OS	Operating System
OSP	Open Scanner Protocol
OWASP	Open Web Application Security Project
PaaS	Platform as a Service
PPA	Personal Package Archive
RAM	Random Access Memory
REST	Representational State Transfer
SaaS	Software as a Service
SCAP	Security Content Automation Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VCS	Version Control System
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language

1 Introduction

1.1 Motivation

Every day more and more services are brought online. Services that may contain sensitive information of the people using them. Passwords, social security numbers, credit card details, classified information regarding their health. Any information that can, and with a great chance, will be used against them if gotten into the untrustworthy hands of a wrongful person.

While a global pandemic has shoved people back into their comfy houses, neither cybercriminals nor their malicious actions are held back by it. Instead, during the year 2020, cybercrime has been lifting its head. Society must adapt to new ways of doing things; everything has to be done online. In a scenario like this, society is at its weakest. As Europol points out in their Internet organised crime threat assessment IOCTA 2020 (2020, 13): “many individuals and businesses that may not have been as active online before the crisis became a lucrative target.” (ibid.)

As the systems, processes, and workflows of the physical world are suddenly bound to shift into the digital one, the reliability and security of the rapidly developed functionality-first services may be questioned. This study was done to reduce the issues mentioned above within the Client organization.

The object of this thesis was to design, develop and implement a cost-efficient security testing solution with automation capabilities. The process included a dedicated server for the testing software and designing and developing a custom tool to interact with it remotely without using the graphical user interface.

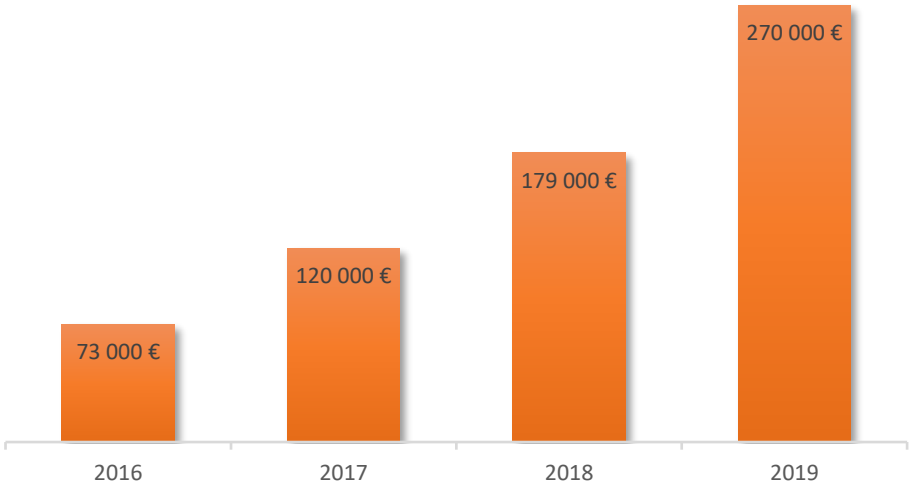
1.2 Client

Liveto Group Oy is a Helsinki domiciled company founded in late 2014, with the primary industry being events (Liveto Group Oy n.d.). It provides a platform with the

needed tools for each type of event, whether it is on-premise, virtual, or a hybrid. Event organizers are able to create and manage events and sell their tickets, food & beverages along with other merchandise within the same platform. The cloud business model is SaaS, Software as a Service. (Liveto n.d.)

The company employs around 20 people, most of whom work at the office located in Jyväskylä. The turnover of Liveto has been increasing steadily, roughly 50 percent every year (Table 1). (Liveto Group Oy n.d.)

Table 1. Liveto's turnover from 2016 to 2019 (Liveto Group Oy n.d.)



From the start of the year 2020, the global pandemic brought the Event industry to a near shutdown via restrictions. The Finnish Government (Government 2020) declared in a press release that "Public gatherings are limited to no more than ten persons, and it is recommend [*sic*] to avoid spending unnecessary time in public places." This restriction led to remarkable changes in the characteristics of the industry. Due to these reasons, Liveto started to develop a solution for migrating live events online.

As Liveto's cloud infrastructure began to grow in terms of server and service quantities, and developers' resources were concentrated on functionalities, concerns regarding cybersecurity started to rise. Fortunately, due to the company's growth,

resources allocated to the enhancement of cybersecurity were more openly available as well.

2 Research Design

2.1 Research Problem

The research problem studied in this thesis was how to perform agile security testing of cloud services. Prior to this thesis, the Client's infrastructure was not tested regularly with security scanners or penetration testing tools, making it prone to potential vulnerabilities and attacks. Due to the agile nature of development, the Client's staging and production environments could be updated multiple times a day; thus, there should be a sensible way of implementation for time-consuming security testing other than on every change of the codebase. Lastly, the implementation's expenses should be as low as possible.

2.2 Research Question

This thesis aims to provide answers to two research questions, which are derived from the research problem. The questions are as follows:

1) How to implement cost-efficient security testing for cloud services?
2) How to automate the implemented security testing in an agile environment?

Measurement

The concept of cost-efficiency is very vague when not defined appropriately. Therefore, the cost-efficiency in this thesis was measured with two resources: money and time. These include the price of the used tools and the developer's time consumed when using the construct. Time equals money so, the time of the

engineering work done to achieve a working implementation must be taken into account when measuring the efficiency.

2.3 Research Methods

To answer the research questions, a suitable research method had to be chosen. The two most common research methods are quantitative and qualitative, which are the basis for plenty of other research approaches and trends.

Quantitative research can be chosen when the variables of the research path, like sample sizes, are known, and the results can objectively be measured. The study usually includes surveys to gather research material, and it follows the research plan strictly. (Kananen 2015, 68-70.)

Qualitative research is usually chosen when there is little or no knowledge of the research subject, which leads to new information and questions during the research. With the constantly appearing new questions, the qualitative methodologies may appear to get cyclic in order to get comprehensive answers for the research questions. (ibid., 69-71.)

Often studies in the information technology industry include products developed during the research process. There is no dedicated research method used for these studies, but they use multi-methodological, so-called blended or mixed methodologies instead (Kananen 2012, 19). This thesis strived to provide a solution for a problem by designing and developing; therefore, the suitable blended methodology to be chosen was constructive research.

The constructive research approach is a methodology to provide constructs - concrete solutions for concrete, real-life problems. Unlike the results of a traditional research, a construct is not discovered but instead invented, developed and created. As Lukka (2001) describes, the study results in “innovative constructs” that include “all the human-created artifacts” which may share the base with something existing

but still provides new value, like an artificial language, the Morse code, for example. (ibid.)

The solution created from a construct can be evaluated using a three-level market test. The levels are defined as follows:

1) The solution works in the subject organization.
2) The solution has been implemented in multiple organizations.
3) The organizations using the solution are objectively doing better than the organizations that are not using the solution.

(Ojasalo, Moilanen & Ritalahti 2015, 68.)

The study aims to fulfill the first level, and at the time, the solution will only be available for the Client organization. However, the implementation shall be done with such quality in mind that it could be implemented in other companies as well.

Research material

The study's research material was primarily gathered from the documentaries of the used tools and technologies, blog posts, and previously done research and publications regarding the subject of security testing and automation solutions.

3 Theory

The theory chapter reviews the concepts, tools, and technologies that are needed for the study, or are already in use within the subject environment, thus restricting other technology choices.

3.1 DevOps

There is no exact definition for DevOps. There are very different opinions on what it comprises but at its core, it is a software development workflow philosophy and a

mindset bringing development-oriented people closer to operations workers, including system, network, and other types of administrators and IT professionals. DevOps has evolved from the “agile” development model, and it aims to enhance the ability to deliver higher quality services with a nimbler pace. (Abildskov 2020; Asay 2017; Mueller 2010; Sharma 2017, xxviii.)

3.1.1 Continuous Integration, Delivery, and Deployment

For a philosophy, DevOps includes a lot of technical tools and practices that ought to help with the process of enclosing the gap of IT departments. The most desired approach must be the automation of all the possible procedures. These procedures usually include merging the new changes to the code base, building the code, and deploying the application into the testing and production environments. The automation can be achieved by applying few practices called Continuous Integration (CI), Continuous Delivery (CD) and Continuous Deployment (CD), most commonly bundled together and referred to as CI/CD. Delivery and Deployment are often mistakenly used interchangeably and overlapped depending on the context, and identical abbreviations do not make it easier.

The CI/CD (Delivery) process intends to integrate committed code into the version control, test and build the software, and finally deliver a new release to be deployed by the Operations staff. CD (Deployment) takes the process one step further than Continuous Delivery and proceeds to deploy the software onto the servers without any human interaction. These automated procedures are mostly called pipelines. (Sharma 2017, 16-18.)

3.1.2 DevSecOps

DevSecOps intends to include the aspect of security within the development process; for example, assessing and acknowledging the threats and risks, and learning secure practices for coding are actions that could be taken. Traditionally security and security scanning is concentrated on the finished application in the production environment and is wholly separated from the development process. DevSecOps

suggests that the security could be automated as well, utilizing the existing CI/CD pipelines. (Matteson 2017.)

Viitasuo (2020) developed a tailored security testing pipeline containing static and dynamic analysis tools and a penetration testing tool. The security testing was integrated into the subject organization's deployment pipeline. Viitasuo did not state the time the pipeline takes to complete, but it is most likely a very notable amount.

Depending on the tools used, security testing can be somewhat time-consuming, meaning that it may not be ideal to be performed inside the CI/CD pipeline – not on every code change. Of course, it depends on the subject organization's business's nature. This thesis aimed to get the best of both worlds and implement security testing to the development life cycle without affecting the pipelines' processing times and developers' workflow significantly. An alternative for the straightforward pipeline solution was to be implemented.

3.2 Vulnerability Scanners

3.2.1 Requirements and Comparison

Vulnerability scanners endeavor to expose any underlying flaws and weaknesses in the subject systems. A suitable scanner tool has to be chosen for the project to succeed. Starting from the most critical one and considering the study's goals, The requirements for the vulnerability scanner are as follows:

1) The vulnerability scanner chosen shall be open-source.
2) The tool must be able to be self-hosted on-premises / on virtual hardware.
3) Has the ability to schedule security scans, automate the tool and communicate with it programmatically.
4) Preferably free of charge.
5) If a stripped-down version of a commercial tool, minimum restrictions for the vulnerability scanning.

The four tools chosen for the comparison in Table 2 are the ones that were most commonly met on different lists of top-rated vulnerability scanners, both free and paid versions. The only open-source tools are Greenbone Vulnerability Manager (GVM) and Zed Attack Proxy (ZAP), which is provided by the Open Web Application Security Project (OWASP). Both are self-hosted, automatable, and free of charge (OpenVAS n.d; OWASP Zed Attack Proxy n.d).

Table 2. Vulnerability scanner comparison

Tool	Open-source	Self-hosted	Automation	Additional info	Price / Year
Greenbone Vulnerability Management / OpenVAS	Yes	Yes	Scheduled scans, documented management protocol / API	The most used open-source vulnerability assessment tool	0 €
InsightVM	No	Yes	Scheduled scans	$25/year/asset	~ 500 €
Nessus Professional	No	Yes	Scheduled scans	The de-facto vulnerability assessment tool	3 368 €
OWASP Zed Attack Proxy	Yes	Yes	Documented API	Mostly web vulnerabilities	0 €

As “Proxy” in the name implies, ZAP is a tool designed for intercepting traffic between the user’s browser and the web application, and the security testing utilities are mainly built on top of this feature. GVM, on the other hand, is more of a framework and provides a broader spectrum of functionality in terms of network and system security testing, though lacking in the web application area.

Even though the Client’s product is a web application, the infrastructure in the background is something to be taken into account as well, and in the best scenario, both tools would be implemented. As a result of the comparison, GVM was chosen to fulfill the study’s requirements.

3.2.2 Greenbone Vulnerability Management

Greenbone Vulnerability Management, GVM, is a comprehensive open-source framework for governing a company's cybersecurity domain. The most critical functionalities include the scanner tool with different intrusion levels and the CVSS (Common Vulnerability Scoring System) ratings for every scanned host. GVM was formerly widely known as OpenVAS (Open Vulnerability Assessment System), which was included in a default "hacker-distro" Kali Linux - explaining a part of the tool's popularity (Ansari & Najera-Gutierrez 2018, 64.) The company behind the vulnerability scanner, Greenbone Network, was brought to the name due to rebranding, and around that time, it was excluded from Kali's default toolbox. OpenVAS was forked from the code base of Nessus, a popular vulnerability assessment tool by Tenable, as it went commercial. In contrast, the starting price point for Nessus is 3368 € per year, while GVM is free (Nessus professional n.d.) OpenVAS (Open Vulnerability Assessment Scanner) is still the default scanner module used in GVM. (OpenVAS n.d.)

GVM uses a community feed to update the NVTs (Network Vulnerability Test) and the SCAP (Security Content Automation Protocol) data on a daily basis. This ensures there are always the latest tests available, and newly discovered vulnerabilities can be found as soon as possible. According to the CEO of Greenbone Networks, Dr. Jan-Oliver Wagner (2019), there are well over 50 000 NVTs for GVM to use, and the number is ever-growing.

GVM offers a graphical user interface known as Greenbone Security Assistant to add target groups, create tasks, and handle the configurations of the vulnerability scans via a web browser. After the scans, GVM provides the user with a scan report including CVSS ratings, the relevant CVE records, and mitigation solutions for the possible vulnerabilities.

GSA uses a management protocol GMP (Greenbone Management Protocol) to communicate with the GVM internals, GVMD (GVM daemon) to be precise. GVMD is

the brain of the system, controlling the scanners, databases, and processing of the data.

The basic topology of GVM includes all the services installed on the same host or container, as seen in Figure 1. This is the configuration achieved by installing GVM using the official repositories or compiling the application from the source code without modification.

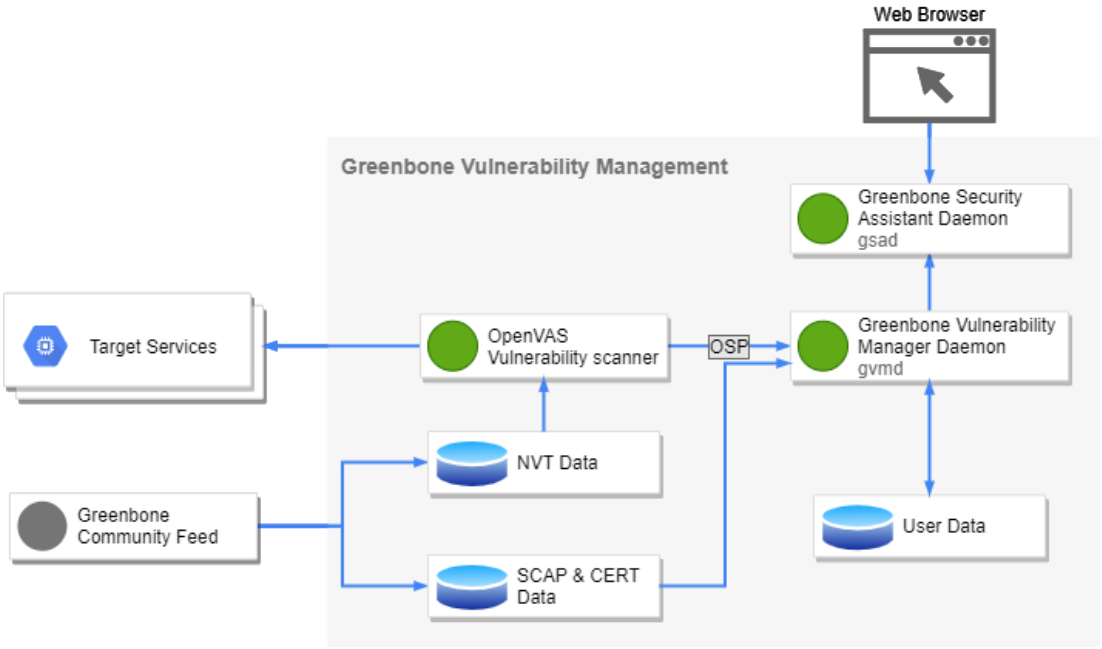


Figure 1. Greenbone Vulnerability Management basic architecture.

3.3 Tools

3.3.1 Google Cloud Platform

GCP is a Google's product providing cloud computing solutions in the forms of IaaS, PaaS, CaaS, and FaaS (Infrastructure, Platform, Container, and Function as a Service). Among many other services, GCP includes Compute Engine, Kubernetes Engine, and Cloud Storage, which are in the Client's use. (About Google Cloud services n.d.)

Compute Engine & Kubernetes Engine

Google Compute Engine provides an IaaS for hosting virtual machines. Google provides all the resources, but there is no administering included; the client is in total control of the VMs and the scaling of their resources. With Google Kubernetes Engine, the client is able to run their containerized applications without having to administrate the host systems as it does in Compute Engine. Kubernetes Engine does use and automate Compute Engine instances to run the containers. (ibid n.d.)

3.3.2 CircleCI

CircleCI is a versatile DevOps tool for running CI/CD pipelines. It provides an opportunity to run the integration and delivery "jobs" in dedicated VMs or Docker Containers. CircleCI allows the developer to access the pipeline with an SSH connection to ensure efficient debugging of jobs. The tool claims to be fast due to allowing parallel runs for workflows and caching unchanged information, like dependencies and container images of prior tasks. The Client uses CircleCI with most of their code repositories to run the tests and perform the deployment. (About CircleCI n.d.)

3.3.3 Nginx

Nginx is open-source web server software, which is famous for its speed and lightweight resource usage. Among Apache, Nginx is the most popular tool used for web serving, load balancing, and reverse proxying. (What is NGINX? N.d.)

Reverse Proxy

A reverse proxy resides in front of a variable number of servers, acting as the endpoint that a client wants to connect to; it is considered to provide better security. When a reverse proxy is implemented, the client will not directly contact the server providing the requested service. Reverse proxy listens for the clients' requests, catches them, and then requests the correct server behind the proxy. From the client's perspective, the requested service is provided by the reverse proxy. (What Is A Reverse Proxy? N.d.)

As the reverse proxy is the only server that the clients will contact directly, it is efficient to use it as the SSL/TLS terminator. Meaning that HTTPS connections are established between the proxy server and the client, rather than separately with the actual endpoints. (What is NGINX?. N.d.)

3.3.4 Python

Python is a high-level interpretable programming language used often in automation-scripting, data-analytics, machine learning, and even web applications. It is an object-oriented language but can be used for procedural and functional programming as well. Among the information and cybersecurity scene, Python is a prevalent language. (General Python FAQ n.d.; Seitz 2014, foreword.)

Besides Python's standard library, there are three additional modules needed to communicate with GVM. "Gvm-tools" is used to establish the connection and transmit the commands between the client and GVM's management daemon (Gvm-tools n.d). The other libraries, "icalendar" and "pytz", are used to create schedules for GVM tasks with the standard internet calendaring system (ICalendar n.d).

At the time of the study, the latest Python version available natively for the development environment was 3.8, which was used when developing and testing the tool. The tool includes syntax and functionality not able to be run on preceding versions of Python 3. The most essential newly introduced functionalities in Python 3.8 are the Walrus operator and Positional-only parameters. (What's New In Python 3.8 n.d.)

3.3.5 Docker

Docker is an open-source software platform that allows a quick and efficient way for programmers to create an isolated application bundled with all of its dependencies in a container and consistently deliver and deploy it in any environment with a Linux kernel. Containerization has become more and more popular for the reason that the very container running on the developer's machine can be deployed, for example, on

the production server, without worrying about the differences in the hardware or the environment. (What is Docker? N.d; Why Docker? N.d.)

Before the era of containerization, a virtual machine (VM) was the industry's go-to technology for isolating a service from another. Figure 2 shows the differences in the abstraction of containers and VMs within a host system. Every VM has a dedicated guest operating system (OS) with a kernel installed on the hypervisor-provided virtualized hardware. Whereas containers only virtualize the OS and share the host system's kernel. Therefore containerization is a lightweight solution compared to VMs. (Containers vs. virtual machines 2019; What is a Container? N.d)

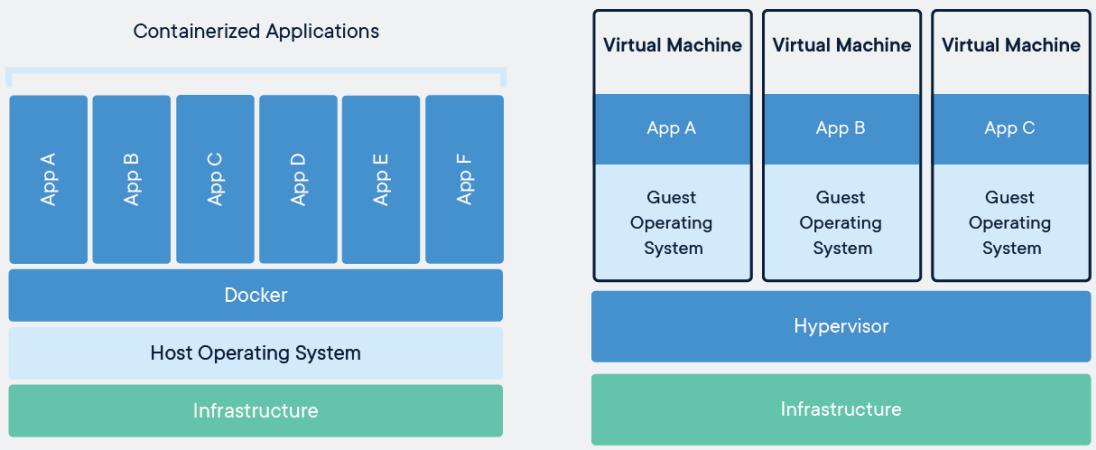


Figure 2. Containers compared to virtual machines (What is a Container? N.d.)

The concept of Docker comprises three essential elements that can be seen in Figure 3. Docker Client provides the interface for the user to communicate with the Docker daemon (dockerd), which is listening to a REST API (Develop with Docker Engine API n.d; Schenker 2018, 17). The two services mentioned above are often, but not necessarily, located on the same host OS. The third service is Docker Registry, remote storage for the Docker Images, which can be self-hosted. However, by default, Docker uses Docker Hub, a public registry maintained by Docker themselves.

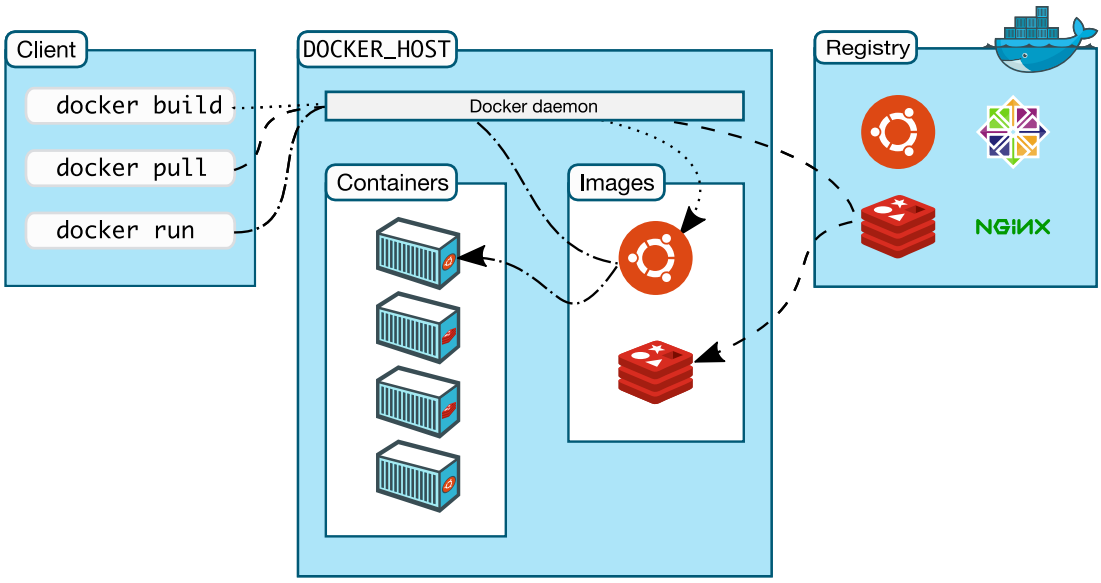


Figure 3. Docker architecture (Docker overview n.d).

Docker Images

The Docker overview documentation succinctly summarizes a Docker image as "a read-only template with instructions for creating a Docker container" (Docker overview n.d). Images are composed of layers of which the first one is a Linux distribution, for instance, Ubuntu, Debian, or Alpine Linux for bare-bones builds. The Docker images with only the first layer are called base images, and other images have the opportunity to use them as parent images. Frequently the base is an official image pulled from Docker Hub. (Schenker 2018, 61-63.)

On top of the base layer resides the needed software. For example, if the container should act as a web server, the software could be Nginx. When using some of the most popular software in their specific areas, an official image can be found again on Docker Hub more often than not. These images are primarily created on top of a base image. That is the case with Nginx as well – by pulling the Nginx image from the official registry, one can create a container with a fully functioning web server running on Debian.

Dockerfile

Dockerfiles are the blueprints for building the Docker Images. As seen in Figure 4, the instructions are very close to what one could run on the command line to configure a service. The containers can be fully configured using the instructions within a Dockerfile, and one does not need to access the container for configuring or starting a service. (Dockerfile reference n.d.)

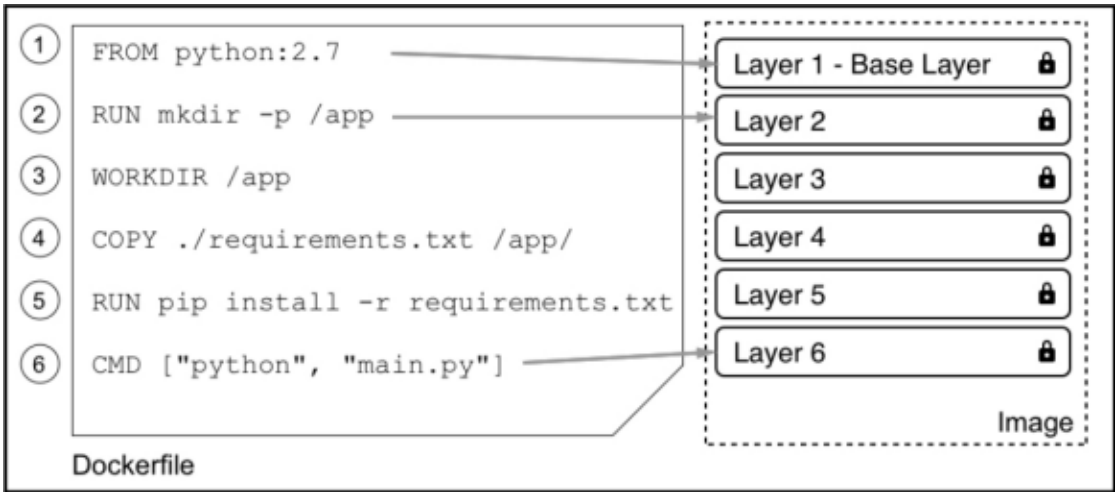


Figure 4. Dockerfile creates layers to Docker Image (Schenker 2018, 69).

Docker Compose

As the Dockerfiles instruct the build of a single image, Docker Compose's primary use is to run and configure services that may contain multiple containers with interdependencies. With a YAML configuration file, one can define the containers, or services, included in the application with their images, ports, volumes, and other options Docker provides for the container configuration. While Docker Compose is a good tool for ensuring identically configured development environments for all the developers and different machines, it can be used in automation and production environments as well. (Overview of Docker Compose N.d.)

4 Implementation

4.1 General

Most of the client's services, subjects of security testing, reside within Google's cloud infrastructure GCP, and that is where the security testing complex was deployed. The deployment and security testing flow is illustrated with six steps in Figure 5. Steps one, two, and four are part of the fundamental development and deployment path; the developer commits their code into VCS, which then triggers the CI/CD pipeline. The pipeline does the necessary unit and integration testing before deploying the compiled application onto the server. Steps three, five, and six form the idea of security testing utilizing the CI/CD pipeline. After the pipeline has successfully tested the new change in code, in step three, it appends the relevant service's FQDN or IP address to a register which is regularly fetched by the security testing server. In step number six, all the fetched hosts are put under security testing at predefined times, for example, every night at 22:00.

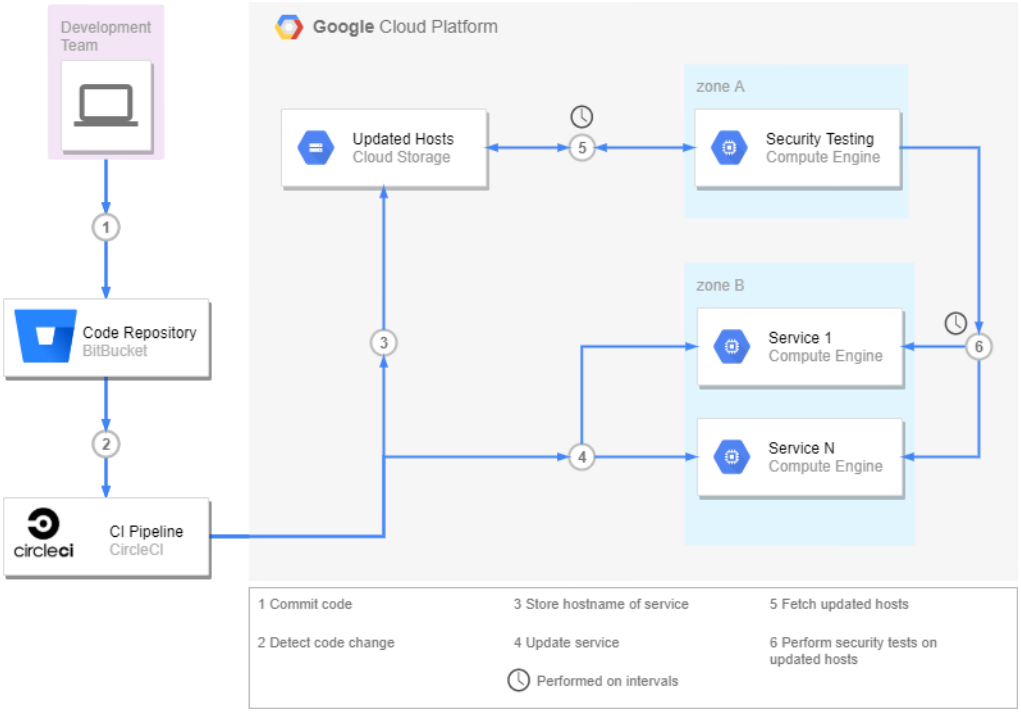


Figure 5. Security testing flow topology.

4.2 Greenbone Vulnerability Management

4.2.1 General

For this thesis, GVM was separated into more minor services in their dedicated containers rather than the whole complex being inside one. This decision was made with the future and scalability in mind as the containerized testing services could be deployed into a Kubernetes cluster later on. Kubernetes would enable expanding the processing capacity by manipulating the number of containers running the heavy load automatically, for example, during the security testing process. This kind of implementation would be more cost-effective since the resources can be brought to the bare minimum when the service is inactive. However, during the thesis-related development, the security testing service was deployed and run within GCP Compute Engine virtual machine with no auto-scaling capabilities.

At the time of starting the study and during the development, the latest stable version for the security testing tool was GVM 11, and it was used for the project. A newer version was released shortly after the end of the development process of this thesis.

4.2.2 Containerization

There are no official Docker Images provided for GVM 11, and to create such images, one should obtain thorough knowledge on the source code of the project in question. Fortunately, an active GitHub user Mohammad “Admirito” Razavi (2021), has achieved to do so and maintains a peer-reviewed code repository “gvm-containers” including a set of unofficial GVM images. Based on the official Greenbone Source Edition (GSE) open-source project, the source code of the separated services is also hosted by Razavi in a Personal Package Archive (PPA). The development of this study is based on the project in question.

As seen in Figure 6, the containerized application topology is slightly different from the basic GVM configuration. GVM-containers project provides five containers: gvmd, gsad, gvm-postgres, openvas, and in addition to the services included in GVM,

the containerized application utilizes a Redis-container to share a Docker Volume involving a socket connection between openvas and gvmd containers. There are no listening ports bound to the host, disallowing any incoming traffic. The incoming traffic includes web browsers, so the user interface cannot be accessed. The application must have a reverse proxy as the gateway for incoming traffic. However, the traffic initiated from the inside is possible.

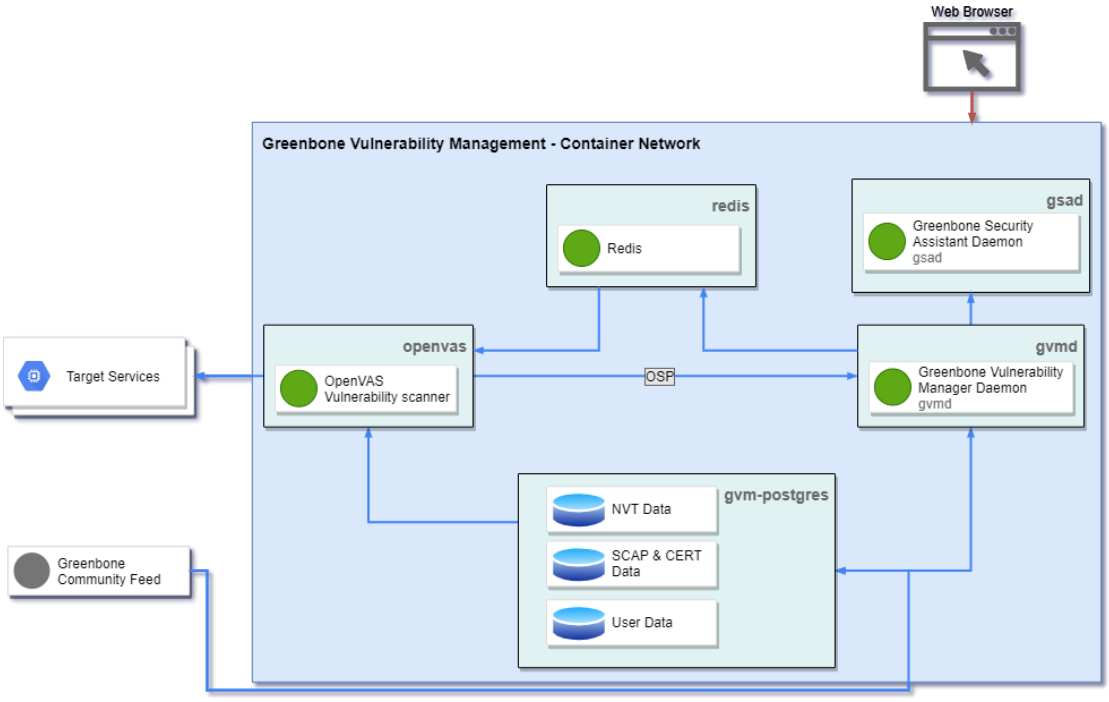


Figure 6. Greenbone Vulnerability Management containerized architecture.

4.2.3 Configuration

The GVM environment is deployed using docker-compose since the containers are interdependent and correct order for the service start-ups is a necessity. The application's compose file (Appendix 1) comprises the five services mentioned earlier with their needed volumes, environment variables, and configuration. However, it is still possible to create and manage the containers separately, though it would be quite a bit more cumbersome process.

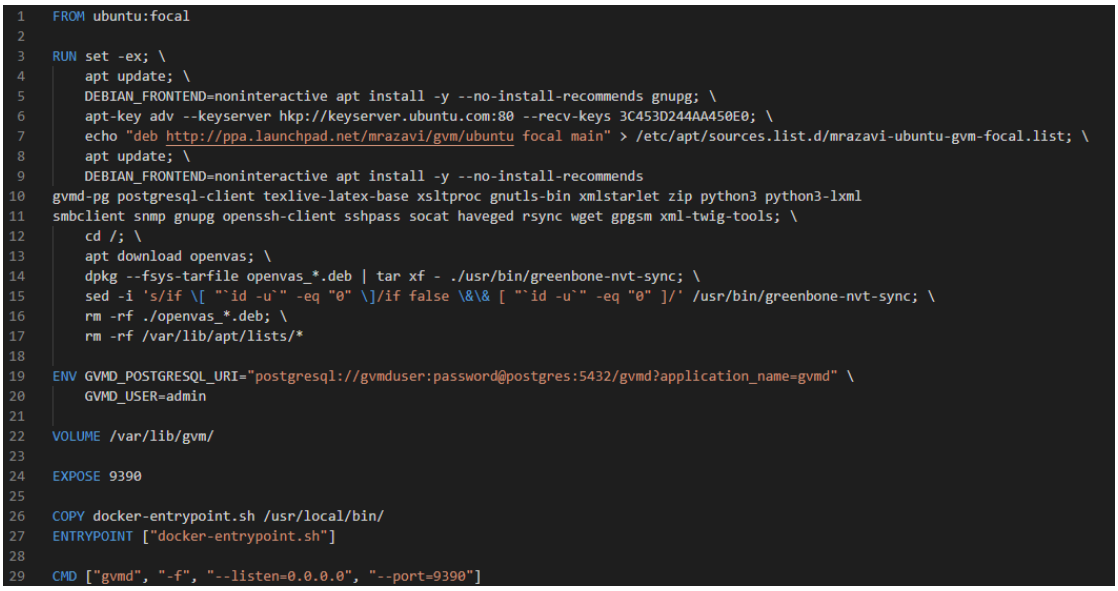
All of the needed images could be pulled from Razavi's Docker registry repository. However, some modifications needed to be done to the images to make them work in a suitable manner for the case. For every service needed, a Dockerfile was also provided; therefore, it would have been unnecessary to configure the images from scratch. In appendix 1, it can be seen that only gvm-postgres, the database, is pulled from the registry by declaring the container's image with "image: admirito/gvm-postgres:11" in line 15. Other GVM containers' images are instead created locally using the "build" command following with the relevant path to locate the corresponding Dockerfile. Redis uses the official image from Docker Hub.

The gvmd container's image was built using a Dockerfile seen in Figure 7. The difference to the provided image lies in the container's logging and the tools that APT installs. The image logged everything in the container's /dev/stdout, which in this case, would print logs to the text terminal. Although the terminal can be read outside of the container, the outputs are not permanent. By default, GVM saves logs in a predefined file, and that file can be shared via a Docker volume, hence the volume "logs" in the compose-file (Appendix 1). The logging functionality was changed for all the images that were built locally to achieve a centralized logging feed in the Docker Volume providing information on all the components of GVM.

With the provided Docker image, GVM's PDF-report generating exhausted the whole system with its actions. The reason appeared to be the lack of gpgsm and xml-twig-tools, causing the generating process to loop infinitely when parsing XML (Extensible Markup Language), and use all memory available. The problem mentioned has been treated in the docker-compose file as well by setting a memory limit for gvmd-service. However, after the proper installations, it is not a necessity but a preventative measure.

The Dockerfiles of other services follow a similar structure with gvmd's: Their base image is Ubuntu 20.04 Focal Fossa, Razavi's repository is added as APT's source, the needed tools are installed, initialization scripts are run from a docker-entrypoint.sh file on the containers start-up, and the service is started after the script finishes. ENVs purpose in the GVM's Dockerfiles is mainly to inform the user, while they are

not used during image building, only in the entrypoint script; thus, they can be overridden in the docker-compose file.



```
FROM ubuntu:focal

RUN set -ex; \
    apt update; \
    DEBIAN_FRONTEND=noninteractive apt install -y --no-install-recommends gnupg; \
    apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 3C453D244AA450E0; \
    echo "deb http://ppa.launchpad.net/mrazavi/gvm/ubuntu focal main" > /etc/apt/sources.list.d/mrazavi-ubuntu-gvm-focal.list; \
    apt update; \
    DEBIAN_FRONTEND=noninteractive apt install -y --no-install-recommends
gvmd-pg postgresql-client texlive-latex-base xsltproc gnutls-bin xmlstarlet zip python3 python3-lxml
smbclient snmp gnupg openssh-client sshpass socat haveged rsync wget gpgsm xml-twig-tools; \
    cd /; \
    apt download openvas; \
    dpkg --fsys-tarfile openvas_*.deb | tar xf - ./usr/bin/greenbone-nvt-sync; \
    sed -i 's/if \[ "`id -u`" -eq "0" \]/if false \&\& [ "`id -u`" -eq "0" ]/' /usr/bin/greenbone-nvt-sync; \
    rm -rf ./openvas_*.deb; \
    rm -rf /var/lib/apt/lists/*

ENV GVMD_POSTGRESQL_URI="postgresql://gvmduser:password@postgres:5432/gvmd?application_name=gvmd" \
    GVMD_USER=admin

VOLUME /var/lib/gvm/

EXPOSE 9390

COPY docker-entrypoint.sh /usr/local/bin/
ENTRYPOINT ["docker-entrypoint.sh"]

CMD ["gvmd", "-f", "--listen=0.0.0.0", "--port=9390"]
```

Figure 7. Modified Dockerfile for gvmd.

After declaring the images in the docker-compose file, environment variables and volumes must be configured for the containers to function. Volumes are used to persist the data and share it between the other services. Environment variables can set information for the service, including usernames, passwords, file paths, port numbers, hostnames, or anything a user traditionally wants to change in system configurations. As mentioned, possible environment variables can be written in services' Dockerfiles to inform the user configuring the system on available environment variables.

Most service sections include ”depends_on” and ”restart” options, ensuring that the containers are started in the correct order and restarted if they appear to be stopped for any unintentional reasons, such as a fatal error in the container or a restart of the host system. In this case, gvm-postgres is the first container to be brought up, following by gvmd and openvas. Redis is started after the openvas container is

successfully started, and gsad is run when gvmd is up. Each container's entrypoint and command option handle the proper connecting with the application.

Options not seen on every service are ”command,” ”sysctls,” and ”privileged.” ”Command” is used to override the default commands run on the image start-up, as with the Redis service, instead of the default TCP configuration, the service is defined to use Unix sockets for better performance. The sockets are located in run-redis volume, which is shared with openvas and gvmd for their connection. ”Sysctls” with ”privileged” options, one can change kernel parameters within the container. In openvas and Redis containers, ”net.core.somaxconn” changes the number of socket connections that can be backlogged. The default value for it is 128, while even the start of Redis service could require over 500.

The containers can be brought up using the command “docker-compose up” (Figure 8). The application would be usable as it now is by opening a port for HTTP traffic to the gsad container, but it was decided that some kind of proxying should be used in between the Internet and GVM.

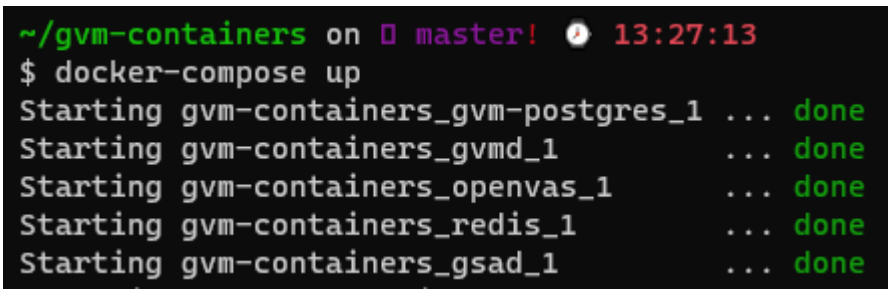


Figure 8. The containers are started with a docker-compose command.

4.3 GCP Compute Engine

4.3.1 General

During the development process, a GCP Compute Engine instance separated from the client's infrastructure was created for the service to take place on. Should there

be problems, it would not affect the client's networking while being configured in the development phase.

4.3.2 Configuration

A compact virtual machine (Figure 9) was deployed on Debian 10 image with one virtual CPU core, 1.7 GB of RAM, and 25 GB of hard drive space. It is enough to get everything working, but when starting to run more extensive vulnerability tests with GVM, the resources are added as well. Both HTTP and HTTPS traffic were allowed from the initial system configuration, but to communicate with GVM via GMP it is necessary to create a new firewall rule. Figure 10 shows that allowing any incoming TCP traffic to port 9390 has been enabled for the VM instances that use the network tag "gvm." That can be seen in the VM's configuration under "Network tags."

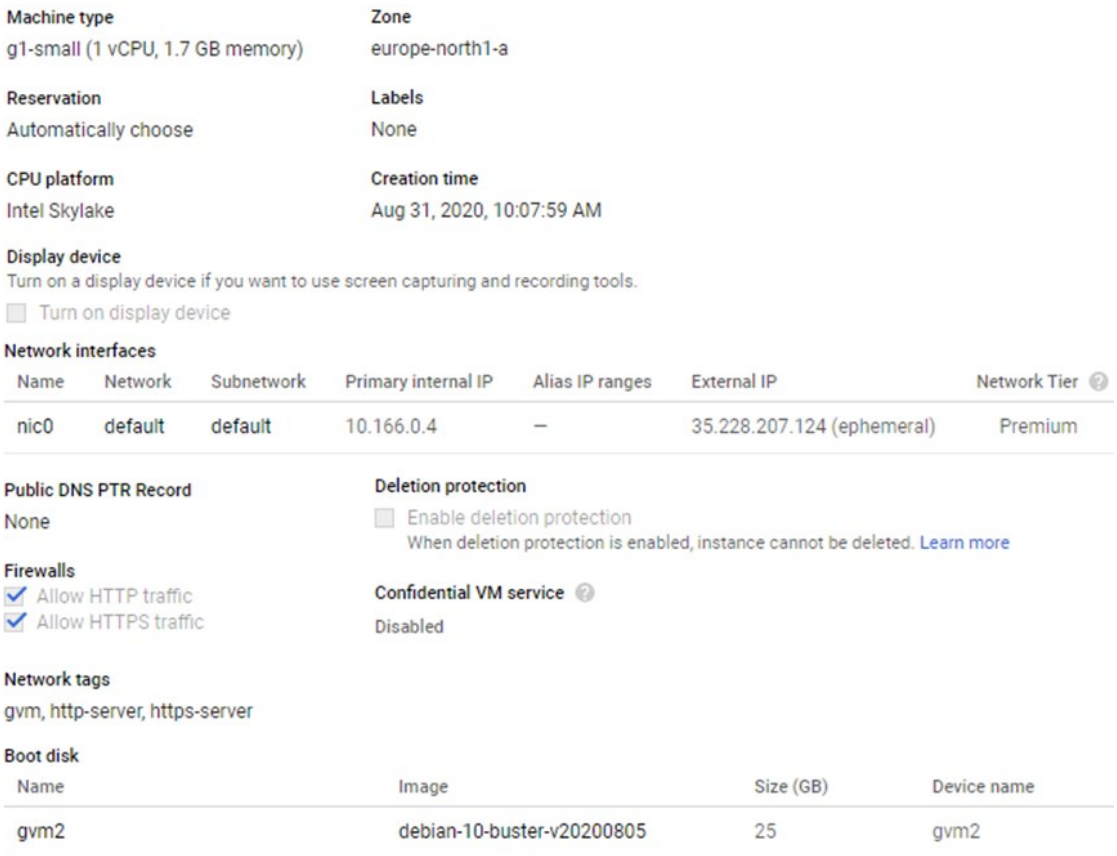


Figure 9. GCP Compute Engine system specifications.

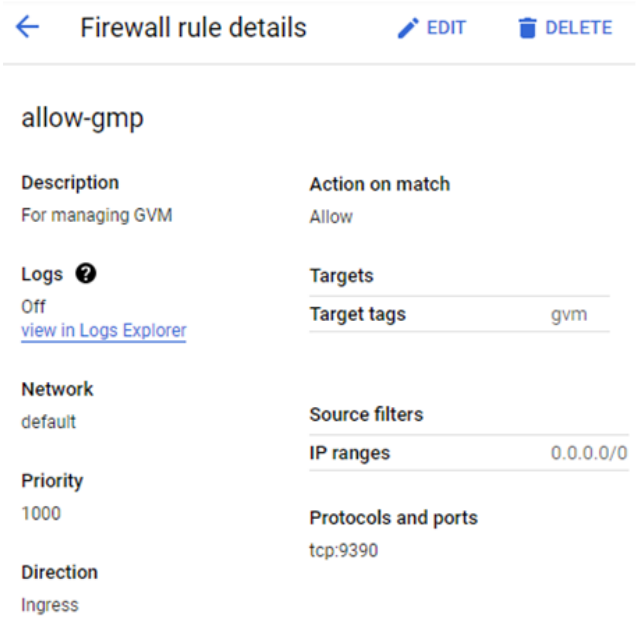


Figure 10. Firewall rule to allow GMP.

After initially configuring the system, the needed additional software was installed on the server, including Docker and Docker Compose.

4.3.3 Domain Name and SSL/TLS Certificate

A free domain name “gvmthesis.tk” was obtained from freenom.com for the development process. The intention was to simulate the production environment as all the routing can be done as it would be with the final product. The ports for HTTP and HTTPS were already opened with the network tags; the only thing missing is the SSL/TLS certificate to achieve a working HTTPS connection.

The certificate was issued using Certbot and Nginx webserver, both containerized. The process requires an Nginx container to be brought up with a specific configuration, and it was decided to be done with Docker Compose again. Figure 11 shows the volume containing the nginx.conf (Figure 12), and the volume that will contain the certificate information after the Certbot container has done its work with issuing the certificate for the server.

```yaml
version: '3.5'

services:
  letsencrypt-nginx-container:
    container_name: 'letsencrypt-nginx-container'
    image: nginx:latest
    environment:
      SERVER_NAME: "gvmthesis.tk"
    ports:
      - '80:80'
    volumes:
      - ./conf/nginx.conf:/etc/nginx/conf.d/default.conf
      - /docker/letsencrypt-docker-nginx/src/letsencrypt/letsencrypt-site:/usr/share/nginx/html
    networks:
      - docker-network

networks:
  docker-network:
    driver: bridge
```

Figure 11. Docker-compose.yml file for Nginx used in obtaining the SSL/TLS Certificate.

```
server {
    listen 80;
    listen [::]:80;
    server_name gvmthesis.tk;
    location ~ /.well-known/acme-challenge {
        allow all;
        root /usr/share/nginx/html;
    }
    root /usr/share/nginx/html;
    index index.html;
}
```

Figure 12. Nginx.conf for obtaining the SSL/TLS Certificate.

When the Nginx container had been brought up, Certbot was run with a shell script containing a Docker Run command (Figure 13). It can be seen in the figure that one of the volumes is the same that is declared in Nginx's compose file. The lines after the image "certbot/certbot" are arguments for the program running inside the container. The arguments include email, agreement for Terms of Service, denial for the provider's emailing list. Webroot path is where the certification data is placed, and it is also the volume for Nginx. The certificate was issued without problems, as can be seen in Figure 14.

```
sudo docker run -it --rm \
-v /docker-volumes/etc/letsencrypt:/etc/letsencrypt \
-v /docker-volumes/var/lib/letsencrypt:/var/lib/letsencrypt \
-v /docker/letsencrypt-docker-nginx/src/letsencrypt/letsencrypt-site:/data/letsencrypt \
-v "/docker-volumes/var/log/letsencrypt:/var/log/letsencrypt" \
certbot/certbot \
certonly --webroot \
--email pyry@email.com --agree-tos --no-eff-email \
--webroot-path=/data/letsencrypt \
-d gvmthesis.tk
```

Figure 13. Certbot.sh.

```
IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/gvmthesis.tk-0001/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/gvmthesis.tk-0001/privkey.pem
   Your cert will expire on 2021-07-24. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot
   again. To non-interactively renew *all* of your certificates, run
   "certbot renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

Figure 14. Obtaining Let's Encrypt SSL/TLS certificate with Certbot.

4.4 Nginx Reverse Proxy

4.4.1 General

Figure 15 illustrates the reverse proxy's purpose in this implementation: to make GVM's web interface available and enable the use of a management protocol GMP. It does also add a layer of security and provide a better, future-proofed user experience. The security comes in the form of access logs and an SSL/TLS-certificate for HTTPS. User experience will be better when multiple different testing tools are running on various unmemorable port numbers. By default, the gvm-containers project runs GSAD in port 8080. With a reverse proxy, one could configure a path to point into a particular port, for example, gvmthesis.tk/gvm. Alternatively, a host-specific approach could be taken, for example, gvm.gvmthesis.tk.

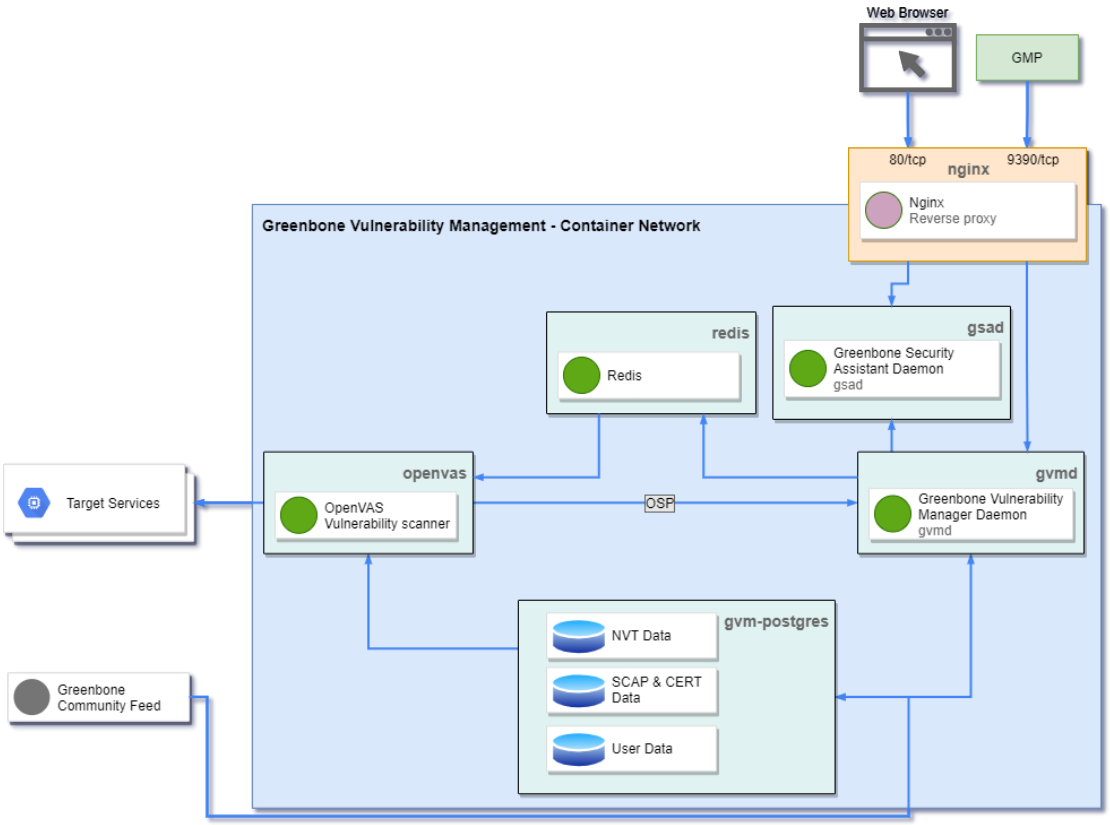


Figure 15. Greenbone Vulnerability Management architecture with reverse proxy.

4.4.2 Configuration – HTTP & HTTPS

Nginx reverse proxy was installed and configured inside a container using an official Docker Image for Nginx. While Docker Compose is a good tool for multi-container applications, it successfully manages single containers with many different arguments as well. As seen in Figure 16, the proxy service does have more than enough configuration to do for a single Docker command.

In addition to Docker Compose's options mentioned earlier, the reverse proxy uses "ports" and "networks." The container's ports have to be bound with the host's ports to access the services from the outside. 80 and 443 are as usual for HTTP and HTTPS, and 9390 is used to communicate with GVM without a web browser, with GMP (Greenbone Management Protocol). The networks option gives the proxy the ability to join the external network that was created at the start of GVM services. When the containers are in the same network, routing can be done using the service names.

```yaml
version: '3.5'
services:
  proxy:
    image: nginx:latest
    environment:
      - SERVER_NAME=gvmthesis.tk
    ports:
      - 80:80
      - 443:443
      - 9390:9390
    volumes:
      - ./conf/conf.d/:/etc/nginx/conf.d/
      - ./conf/tcpconf.d/:/etc/nginx/tcpconf.d/
      - ./conf/includes/:/etc/nginx/includes/
      - ./conf/nginx.conf:/etc/nginx/nginx.conf
      - ./service-not-found.html:/usr/share/nginx/html/service-not-found.html
      - /var/log/nginx/:/var/log/nginx/
      - /docker-volumes/etc/letsencrypt/:/etc/letsencrypt/
    networks:
      - gvm-containers_default
    restart: always

networks:
  gvm-containers_default:
    external: true
```

Figure 16. Nginx docker-compose.yml file.

The docker-compose file shows that there are multiple volumes, many of which are binding a configuration folder. Nginx does not compel one to write separate files for each type of configuration – everything can be put inside nginx.conf. While projects and services become more extensive in terms of configuration lines, it is definitely a good practice to introduce the configuration blocks in their descriptive files. The files can be imported to the appropriate place, as has been done in this implementation. The folders' structure is illustrated with Tree-commands output in Figure 17.

```
~/thesis/nginx on  prod!  21:21:31
$ tree conf
conf
├── conf.d
│   └── default.conf
├── includes
│   ├── proxy.conf
│   └── ssl.conf
├── nginx.conf
└── tcpconf.d
    └── gmp.conf
```

Figure 17. Nginx configurations formatted with Tree-command.

The root of the configuration files is “nginx.conf” (Figure 18) which the Nginx server reads first by default. Other configuration locations are imported there directly and indirectly via other imported configuration files. The broadest configurations, like the HTTP block, are placed there among the process options. The HTTP block defines the format for log messages, location for the access log, request and response compression with gzip is enabled, and the configurations in conf.d folder are included, imported, in the block.

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log          /var/log/nginx/access.log  main;

    sendfile            on;
    keepalive_timeout   3600s;
    gzip                on;

    include             /etc/nginx/conf.d/*.conf;
}

# SSL passthru
include /etc/nginx/tcpconf.d/*.conf;
```

Figure 18. Nginx.conf file.

The only configuration file in the conf.d folder is default.conf file, which contains the Server blocks as seen in Figure 19. The first Server block listens to port 80 and recognizes “gvmthesis.tk” as server’s name. All the traffic coming to port 80 is

insecure HTTP, and this block's duty is to make sure that the traffic is redirected automatically with HTTP status code 301 to HTTPS in port 443 listened to in the other Server block. The second block defines the primary location redirection, the SSL/TLS Certificates, and includes the configuration (Figure 20) for HTTPS. It accepts only TLS versions 1.2 and 1.3 since the older versions are deprecated. The "location /" block includes proxy configuration (Figure 21) and proceeds to redirect incoming traffic to the relevant upstream.

```
upstream docker-gsad {
    server gsad:80;
}

server {
    listen 80;
    server_name gvmthesis.tk;
    return 301 https://gvmthesis.tk$request_uri;
}
server {
    listen 443 ssl http2;
    server_name gvmthesis.tk;

    # certs
    ssl_certificate /etc/letsencrypt/live/gvmthesis.tk-0001/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/gvmthesis.tk-0001/privkey.pem;
    include /etc/nginx/includes/ssl.conf;

    location / {
        resolver 127.0.0.1 valid=30;
        include /etc/nginx/includes/proxy.conf;
        proxy_pass http://docker-gsad$request_uri;
        proxy_redirect off;
        error_page 500 = @error/service-not-found.html;
    }

    location @error {
        root /usr/share/nginx/html;
    }

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log error;
}
```

Figure 19. Nginx default.conf file.

```
ssl_session_timeout 1d;
ssl_session_cache shared:SSL:50m;
ssl_session_tickets off;

ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers 'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:
ssl_prefer_server_ciphers on;
```

Figure 20. Nginx ssl.conf file.

```
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

Figure 21. Nginx proxy.conf file.

In this case, the upstream is GVM, precisely the gsad-container that is a part of the GVM complex. The Nginx container's docker-compose file defines GVM's default network as an external network that allows the Upstream block to connect to the GVM containers using only the services name. It is intended to connect the user with a web interface to GVM, provided by Greenbone Security Assistant service, running in the gsad container's port 80.

After running "docker-compose up" for GVM containers and the Nginx server, it is possible to connect to our service with a browser. In Figure 22, the requested address "http://gvmthesis.tk" redirects to "https://gvmthesis.tk/login" with a visible lock icon in the address bar. This kind of behavior indicates that the redirect from HTTP port 80 to HTTPS port 443 and the SSL/TLS Certificate is working correctly.

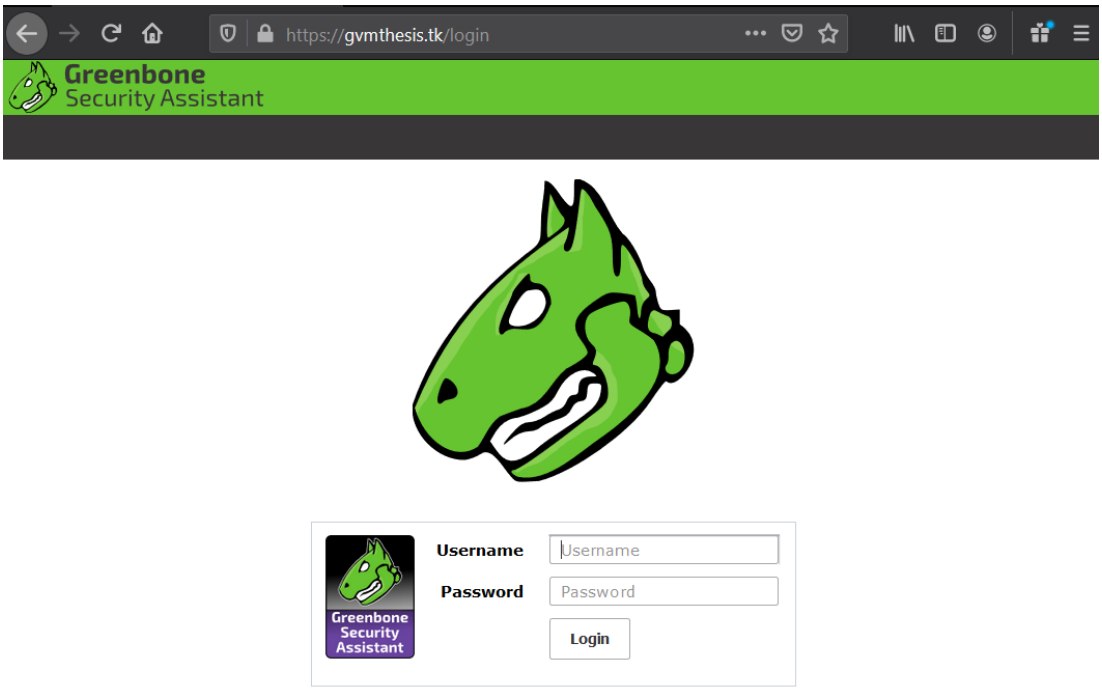


Figure 22. HTTPS is working with the SSL/TLS certificate.

4.4.3 Configuration – GMP Passthrough

Interaction with GVM without using the graphical web interface requires the use of GMP. By default, the GVMD listens to port 9390/TCP, and it could be opened for the traffic from the outside, but the better option would be to use the existing Nginx service to pass the GMP traffic to the gvmd container. This way, all the GVM containers can be kept behind a monitored entity. While every incoming connection flows through a single container, it is easier to monitor access logs when they can be separated. The web interface does communicate to GVMD with GMP as well, so all traffic is logged in the same place on GVMD's end. In Figure 23, GMP's access log has been defined to its dedicated file: gvmd_access.log, separated from the HTTP traffic. The entire configuration is located inside a Stream block, which can be considered similar to the HTTP block, except it forwards TCP connections. The Server block tells Nginx to listen to port 9390 and pass the traffic to the Upstream block's gvmd port 9390.

```
stream {
    log_format basic '$remote_addr [$time_local] '
                     '$protocol $status $bytes_sent $bytes_received '
                     '$session_time';

    access_log /var/log/nginx/gvmd_access.log basic;

    upstream docker-gvmd {
        # gvmd server listening for GMP
        server gvmd:9390;
    }

    server {
        listen 9390;
        proxy_pass docker-gvmd;
    }
}
```

Figure 23. Nginx gmp.conf file.

4.5 GMP Tool

4.5.1 General

Human interaction is, on average, easiest to do with a graphical user interface, but machines and software have a hard time using graphical interfaces. Fortunately, Greenbone provides a gvm-tools library for Python, which is essentially helpful when commanding GVM from, for example, a CI/CD pipeline. Gvm-tools does ship with a CLI tool, but its functionality was very clunky with remote connections. With the library, it was possible to develop a tailored tool for the needs of this project. There are few requirements for the tool to fulfill. The remote user should be able to add new targets and tasks with their scan configurations to GVM, list existing ones, schedule, and start tasks. Greenbone provides technical documentation for GMP, and it is used to ease the tool's development process (Greenbone Management Protocol n.d).

4.5.2 Development

The tool's (Appendix 2) development began with installing the gvm-tools library with the command "pip3 install gvm-tools", which allows Python to import needed functionality. The first thing that needed to be done was to establish a connection to the server running GVM. Gvm-tools provides three ways to connect to a GVM server: SSH, TLS, and Unix sockets. The TLS connection was chosen since it includes built-in security and does not require any additional configuration on the GVM server. It was already configured to listen to port 9390, which is also the default port for gvm-tool's TLS connection. The relevant imports for the connection are located in lines 6, 7, and 8 in Appendix 2 – TLSconnection initializes the secure connection, EtreeTransform formats the responses as XML element trees. Gmp class is used as a context manager for the GMP traffic in line 212.

The Gmp class includes an authenticate method, which takes username and password as arguments and has to be successful to perform further actions. The login function in line 187 tries to authenticate and checks from the response message if it was successful. The response is a byte string containing data in XML format, as it was defined, and the response status has to be retrieved using xpath. The response in Figure 24. Successful authentication response from GVM. has been printed with the pretty_print command imported from gvm.xml to enhance readability.

```
<authenticate_response status="200" status_text="OK">
  <role>Admin</role>
  <timezone>Europe/Helsinki</timezone>
  <severity>nist</severity>
</authenticate_response>
```

Figure 24. Successful authentication response from GVM.

After the authentication was handled sufficiently, the first and the most straightforward feature to implement was retrieving the list of tasks and targets.

Gmp class does include a method to get all data of the targets (Figure 25) and tasks, but the amount of information exceeds what is needed to check the names of existing subjects.

```
<get_targets_response status="200" status_text="OK">
  <target id="50cfbc20-5b71-41ea-ab50-5b28eca49671">
    <owner>
      <name>admin</name>
    </owner>
    <name>google.com</name>
    <comment/>
    <creation_time>2020-09-23T08:37:20+03:00</creation_time>
    <modification_time>2020-09-23T08:37:20+03:00</modification_time>
    <writable>1</writable>
    <in_use>0</in_use>
    <permissions>
      <permission>
        <name>Everything</name>
      </permission>
    </permissions>
    <hosts>google.com</hosts>
    <exclude_hosts/>
    <max_hosts>1</max_hosts>
    <port_list id="c7e03b6c-3bbe-11e1-a057-406186ea4fc5">
      <name>OpenVAS Default</name>
      <trash>0</trash>
    </port_list>
    <ssh_credential id="">
      <name/>
      <port/>
      <trash>0</trash>
    </ssh_credential>
    <smb_credential id="">
      <name/>
      <trash>0</trash>
    </smb_credential>
    <esxi_credential id="">
      <name/>
      <trash>0</trash>
    </esxi_credential>
    <snmp_credential id="">
      <name/>
      <trash>0</trash>
    </snmp_credential>
    <reverse_lookup_only>0</reverse_lookup_only>
    <reverse_lookup_unify>0</reverse_lookup_unify>
    <alive_tests>Scan Config Default</alive_tests>
  </target>
```

Figure 25. Gmp get_targets method’s output for one target.

The relevant data (Figure 26), including the task’s or target’s name and their ID, was parsed from the responses with get_targets and get_tasks functions on lines 130 and 155. The result is far more readable compared to the initial response and is most likely enough for the intended use. Although, a verbosity option could have been implemented to change the specificity of the output. Fetching information on schedules and scanner configurations works similarly to the aforementioned functions.

```
list_targets
[('google.com', '50cfbc20-5b71-41ea-ab50-5b28eca49671'), ('gruyere', 'b6f314ea-572e-4575-ab22-be2e5cb44710'), (
'gvmthesis.tk', '1345b215-8ab5-483c-8b7c-f999006dd704'), ('localhost', '562a1628-94db-4cfe-a514-3cd94057e61f')]
```

Figure 26. GMP tool listing target information.

The next feature to implement was supposed to start existing tasks by giving the task's name as an argument. Gmp does obviously provide a start_task method, but it takes the task's id as an argument. This means it had to be determined which name has which id, and luckily a function to contain both information was created earlier. The get_task_by_name function on line 143 uses the get_tasks function to fetch the tasks' names and IDs, and then compares which of the pairs has the requested name and proceeds to return the corresponding task_id to the start_task function on line 168. Figure 27 illustrates the XML response received from the server and the request for starting a task named "gvmthesis.tk" seen via the web interface.

```
start_task
<start_task_response status="202" status_text="OK, request submitted">
  <report_id>fc114090-d31e-48a8-9715-d6ccd382d929</report_id>
</start_task_response>
```

Figure 27. Starting a task using the GMP tool.

Creating a task requires there to be something as a target first. GVM requires a list of hosts and a name for that target group, and there is an optional argument used to comment on the group, as seen in the function on line 22. The function was designed so that the user does not need to provide a name for the target group. In this case, the name will be the same as the host argument.

Now that there could be added targets to have tasks, the create_task function on line 47 was implemented. GVM requires four arguments: name, config_id, target_id, scanner_id, and optional schedule and comment arguments. The intuitive way to define the targets would be to use the name and to achieve that, the target_id was fetched with the help of existing functionality, pairing targets' name and ID, and returning the matching ID for the name. There was only one option for a scanner at the time: OpenVAS. Its ID is fetched from GVM, and the value's xpath is hardcoded in the tool.

OpenVAS scanner provides eight default scanning configurations to choose from, each of which varies in the types and number of NVTs run, meaning the level of features and intrusiveness. Besides the name and ID, GVM returns much unnecessary data (Figure 28) when requesting configurations. Writing the names nor the IDs to select the configuration is not optimal for the user. Therefore, a Python dictionary was implemented (Appendix 3, line 4) with integers as keys and the configuration names as their values. The get_config_id function in Appendix 2 line 33 takes the dictionary's key from the user and then, with the corresponding value, the scan's name, searches the ID pair for that value and returns it to the create_task function.

```
>>> with Gmp(connection, transform=transform) as gmp:
...     auth = gmp.authenticate(GVM_USER, GVM_PASS)
...     resp = gmp.get_configs()
...
>>> pretty_print(resp)
<get_configs_response status="200" status_text="OK">
  <config id="d21f6c81-2b88-4ac1-b7b4-a2a9f2ad4663">
    <owner>
      <name/>
    </owner>
    <name>Base</name>
    <comment>Basic configuration template with a minimum set of NVTs required for a scan.</comment>
```

Figure 28. The relevant part of the scanner configuration's response in Python shell.

The schedules are iCalendar objects, and they are created in Appendix 2 line 92 create_schedule function, if the requested schedule has not already be found in get_schedule_by_name function on line 69. The Schedules are configured to run at the following midnight and then, for example, daily or weekly.

4.5.3 Development – CLI Tool

As all the intended functionalities have been implemented, the features should be possible to use via terminal as well. Python's standard library provides a handy module for CLI programming: Argparse. With the module, one can use command-line options, known as flags, to achieve the wanted outcome with the program. Argparse can raise errors if one is missing a required parameter and automatically generates a help page, including all the flags and their options if defined. The flag options can be seen in Figure 29 showing the help page.

```
~/thesis on  prod!  9:17:35
$ python3 gmp.py --help
usage: gmp.py [-h] [-a | -A] [-C {0,1,2,3,4,5,6,7,8}] [-t TARGET] [-H HOST] [-n NAME] [-c COMMENT] [-S]
 [-s {now,daily,weekly}] [-lt] [-lT]

optional arguments:
  -h, --help            show this help message and exit
  -a, --add-target      Add a new target
  -A, --add-task        Add a new task
  -C {0,1,2,3,4,5,6,7,8}, --scan-config {0,1,2,3,4,5,6,7,8}
                        {0: 'Base', 1: 'Discovery', 2: 'Full and fast', 3: 'Full and fast ultimate', 4:
 'Full and very deep', 5: 'Full and very deep ultimate', 6: 'Host Discovery', 7: 'System Discovery'}
  -t TARGET, --target TARGET
                        ID or name of the target, required with add-task
  -H HOST, --host HOST  IP address or hostname for new target
  -n NAME, --name NAME  Name for target/task, required with add-target and add-task
  -c COMMENT, --comment COMMENT
                        Add comment for created object
  -S, --start-task      Start task
  -s {now,daily,weekly}, --schedule {now,daily,weekly}
                        Set task schedule
  -lt, --list-targets   List target names and IDs
  -lT, --list-tasks     List task names and IDs
```

Figure 29. Gmp.py help page generated by Argparse.

Figure 30 illustrates the command and output of creating a target using the developed tool, followed by the result seen in the GVM web interface in Figure 31. The flags used in the command are shorthands for –add-target, --host, --name, and –comment.

```
~/thesis on  prod!  11:15:57
$ python3 gmp.py -a -H gvmthesis.tk -n gvm_cli -c 'Hello World!'
login successful

create_target
<create_target_response status="201" status_text="OK, resource created" id="0847464b-3533-4585-a09d-56989e8ca155"/>
```

Figure 30. Adding a target using gmp.py.

Figure 31. Added target in GVM web interface.

Now that the target has been established, a task for that target can be created. Figure 32 shows a command creating a scheduled task using the “Host Discovery” configuration. The GVM web interface shows a new task in Figure 33 with a weekly schedule, and the first run being at the following midnight.

```
~/thesis on  prod!  11:38:40
$ python3 gmp.py -A -t gvm_cli -C 6 -s weekly -c 'Hello Scheduled Task!'
login successful

create_task
<create_task_response status="201" status_text="OK, resource created" id="77d416a6-2564-4604-b39e-d99e31662a64"/>
```

Figure 32. Adding a task using gmp.py.

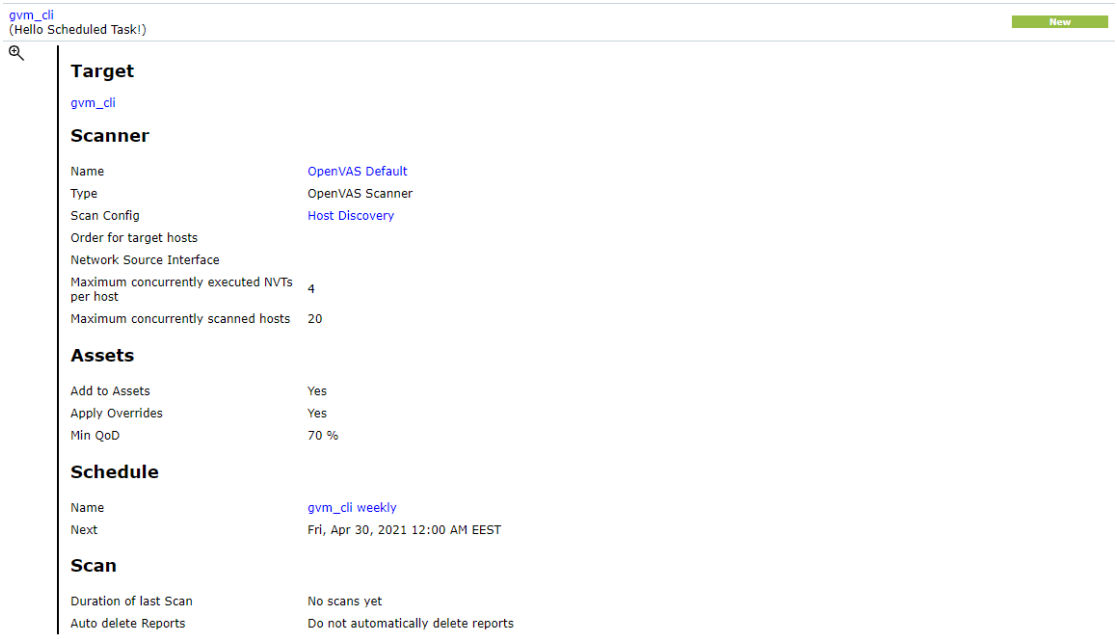


Figure 33. Added task in GVM web interface.

Scheduled tasks cannot be run manually, but a task can be created without a schedule, thus it can only be run manually. Figure 34 and Figure 35 show the command to create a task and start it immediately and how it shows up in GVM.

```
~/thesis on  prod! 🕐 11:58:40
$ python3 gmp.py -A -t gvm_cli -C 6 -S -c 'Hello Started Task!'
login successful

create_task
<create_task_response status="201" status_text="OK, resource created" id="7f515fd0-1c9c-480d-a4bd-265dde2552ec"/>
```

Figure 34. Adding a task and starting it immediately.

Figure 35. The GVM web interface shows the started task as requested.

4.6 CI/CD

4.6.1 General

There was some configuration to do in the CI tool to get the information of the code changes to the GVM server. The client uses CircleCI as their tool for CI/CD, and its configuration files consist of different workflows, which include multiple jobs and steps. The configuration files are written in YAML.

There were two options for the implementation of informing GVM:

1) Download the gmp.py (Appendix 2) tool with its requirements into every CI instance run and individually create a task for every host changed. The tool could also be added to every repository instead of downloading it.

2) Append the hostname of the changed server into a file located in Google Cloud Storage. The GVM server would periodically fetch the file, and the file's hosts would be added to a single target and task with the gmp.py tool.

The second option was chosen to be implemented due to the fact that it could be implemented in the CI with less configuration and without downloading any excess tools. Google's cloud-sdk Docker image is already used in the configuration, and it does provide the necessary tools for accessing Google Cloud Storage. The authentication information for Google's services is already found in CircleCI's secrets, and there is no need to add the GVM server's credentials. Adding the secrets would have been done separately for every repository's CI pipeline. Also, inserting the GMP tool into every repository would lead to unnecessary work when the tool is updated – It would be better to only have the tool in one place, in the GVM server, and update it along with the GVM-container's CI pipeline.

4.6.2 Configuration – CircleCI

There was not much of configuration to do to create a universal block seen in Figure 36. The CI job's name is set-security-testing, and it runs in a Docker container built from Google's cloud-sdk image. Environment variables define the host to scan and the information for authenticating to Google Cloud, which is handled in the first step. The second step handles copying the existing file from Google Cloud Storage and

appending the hostname at the end of the file. The file is then updated to Google Storage using a rsync command instead of cp, copy - this is to prevent overwriting some other host's update if multiple CI pipelines are being run simultaneously. Figure 37 shows the workflow configuration in the same file's end. It states that in order to start the set-security-testing job, the push-to-staging job has to be successfully finished, and that the job is only run when committing code to the develop branch. This way, the code is tested when it is not yet in production.

```
+  set-security-testing:
+
+    docker:
+      - image: google/cloud-sdk:latest
+
+    environment:
+      - GOOGLE_PROJECT_ID: "                "
+      - GOOGLE_COMPUTE_ZONE: "europe-west1-b"
+      - HOST_TO_SCAN: "                    "
+
+    steps:
+      - run:
+          name: Setup Google Cloud SDK
+          command: |
+            echo $GOOGLE_AUTH | gcloud auth activate-service-account --key-file=-
+            gcloud --quiet config set project ${GOOGLE_PROJECT_ID}
+            gcloud --quiet config set compute/zone ${GOOGLE_COMPUTE_ZONE}
+
+      - run:
+          name: Add a relevant host to updated_hosts.txt on GS
+          command: |
+            mkdir tmp && cd tmp
+            gsutil cp gs://                          /updated_hosts.txt .
+            echo $HOST_TO_SCAN >> updated_hosts.txt
+            gsutil rsync . gs://                          /
```

Figure 36. CircleCI job for setting updated host.

```
173 +      - set-security-testing:
174 +          requires:
175 +            - push-to-staging
176 +          filters:
177 +            branches:
178 +              ignore: master
179 +              only: develop
```

Figure 37. CircleCI workflow's configuration.

4.6.3 Configuration – Server

On the GVM server itself, fetching the updated hosts is performed with a Bash script, as illustrated in Figure 38. The script rsyncs the folder containing updated hosts to its docker folder. This is where all the source code relevant to the study is stored. The newly downloaded file is compared with the previous file, and the additional lines are used to overwrite test_hosts.txt. The script is exited if the file is empty.

A Python virtual environment is activated, containing all the requirements for the GMP tool to function. The insides of the file are sorted, and possible redundancies are removed. The remaining lines are concatenated as a string that the GMP tool accepts as an argument, and the tool is used first to create the target, then the task, and finally to start the scanning. The TARGET_NAME variable is timestamped in Figure 38 line 26 to tell apart the previously created targets.

The Bash script is run periodically with the help of Cron. Cron is a built-in feature in Unix systems to run scheduled commands and scripts. The intervals can be set as one will, for example, every day.

```bash
#!/bin/bash

gsutil rsync gs://[redacted]/ /docker/
comm -13 /docker/updated_hosts.txt.old /docker/updated_hosts.txt > /docker/test_hosts.txt
mv -f /docker/updated_hosts.txt /docker/updated_hosts.txt.old


if [ -s /docker/test_hosts.txt ]; then
  echo 'Hosts to test'
else
  echo 'No hosts to test'
  exit 0
fi

. /docker/gvm_venv/bin/activate

# Get updated hosts and format a string
File='/docker/test_hosts.txt'
Hosts=$(sort -u $File)
for Host in $Hosts; do
  NEW_HOSTS=$Host,$NEW_HOSTS
done
NEW_HOSTS=$(echo $NEW_HOSTS | sed 's/,*$//g')


TARGET_NAME=updated_$(date +'%Y-%m-%d')
SCAN_CONFIG=2

python3.8 /docker/gvm/gmp.py --add-target --host $NEW_HOSTS --name $TARGET_NAME
python3.8 /docker/gvm/gmp.py --add-task --target $TARGET_NAME --scan-config $SCAN_CONFIG \
  --comment 'Automatically created task to scan recently updated hosts' \
  --start-task
```

Figure 38. Bash script to fetch updated hosts and start a scanning task.

5 Results

The final product developed and implemented as a result of this thesis process is a containerized vulnerability scanning and management service, capable of being tasked dynamically. Tasking can be either fully automatic utilizing Continuous Integration and Continuous Delivery pipeline or manual via a web interface or a custom developed command-line interface program.

Whenever a developer commits into a branch chosen to be the subject for testing, the CI/CD pipeline requests a scan for a particular host. The requested hosts are used to define new tasks for the GVM with the developed GMP tool. After the tasks have been successfully run, GVM provides vulnerability reports, possible mitigation solutions, and beyond via the graphical web interface, to whom it may concern. Every developer could review the results of the scanning. However, the person

responsible for the information security in the organization would be a better pick for assessing the reports and the further actions that ought to be taken. Figure 39 represents GVM's vulnerability reporting with the data gathered by scanning a vulnerable demonstration server.

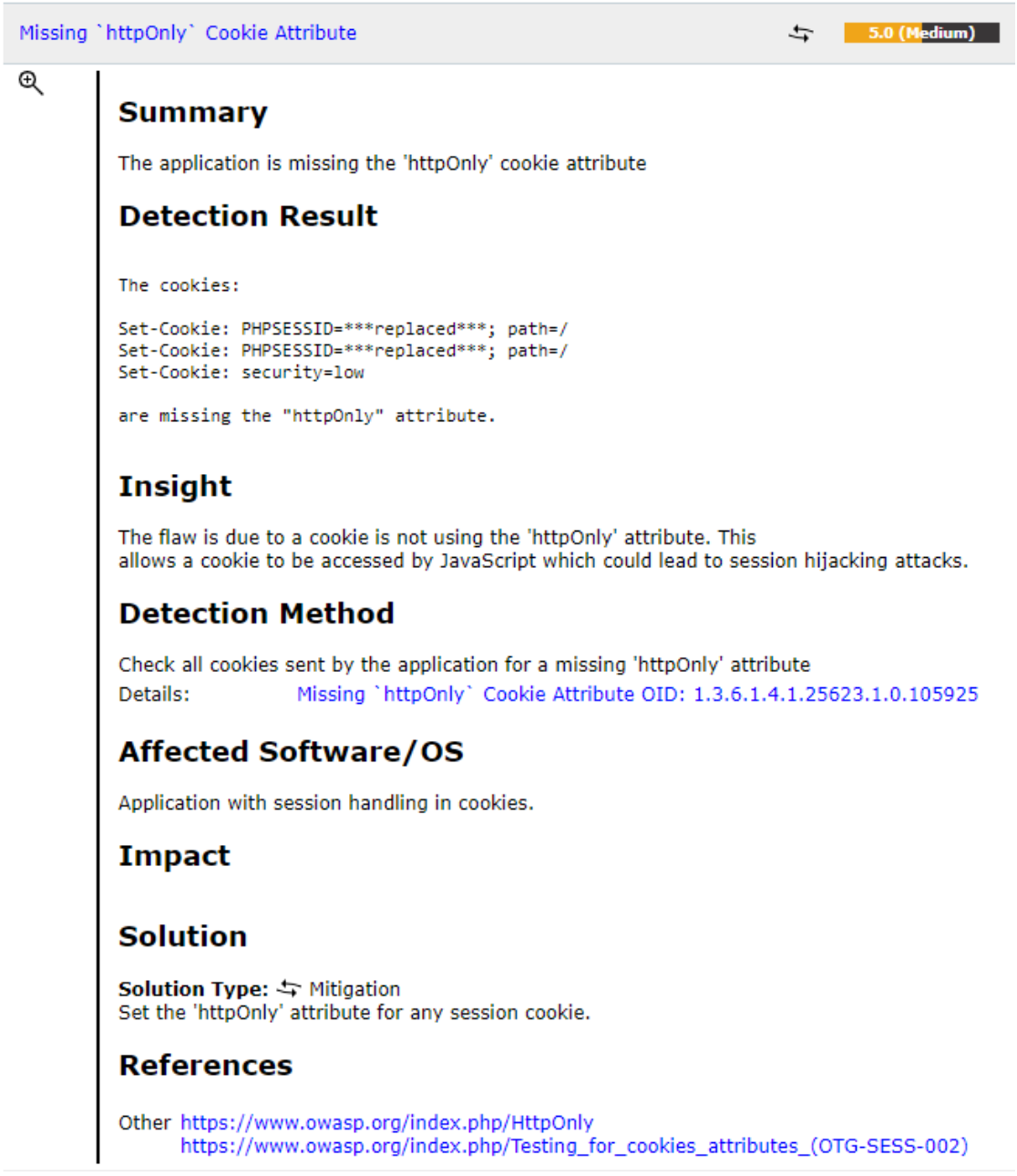


Figure 39. Example of a GVM vulnerability report.

6 Conclusion

The goal set for this Bachelor's thesis was to develop and implement an automated and cost-efficient security testing tool for cloud services without slowing down the developers' normal working flow. The study was to answer two research questions: How to implement cost-efficient security testing for cloud services and how to automate the implemented security testing in an agile environment.

The required cost-efficiency was achieved using platforms already in the Client's infrastructure: CircleCI and Google Cloud Platform; and selecting additional free-of-charge, open-source tools, including Docker, Greenbone Vulnerability Management, and Nginx. The efficiency can be pushed further if GVM is deployed into Kubernetes with automatic resource scaling, hence the already containerized application. The cost of the author's engineering work needed to bring up a running security testing service was, at estimate, half of the cost of purchasing a one-year license for Nessus.

The implementation retained the developers' regular flow by informing the testing tool that the service has been changed, rather than running the time-consuming testing within the CI/CD pipeline.

The set requirements were met with the study's end product, meaning that the construct does provide a solution for the presented problem. The solution passes the first level of the Three-level market test, mentioned in chapter 2.3, which requires that the solution works in the subject organization. When taking the scope of the study into account, the other levels are unachievable at this point. However, the concept of automated security testing utilizing CI/CD technologies could quite possibly be implemented within any organization in the industry providing software services.

6.1 Reliability

GVM includes Quality of Detection metrics providing a value from 0 to 100 % depending on the detection methods. However, the systems are scanned using only one tool, which creates a chance of having false-positive results or undetected weaknesses. Implementing another tool to run with GVM could drastically improve detection reliability.

6.2 Challenges

The biggest challenges occurred while developing the GMP CLI tool. The management protocol's documentation is a bit lacking, and there were not many additional discussions to be found. Many problems were solved by trial and error, and by reading the GVM-tools package's source code.

Also, some of the problems encountered when configuring GVM were beyond the reach of the tool's official documentation and required a significant amount of knowledge in system administration or getting known to the source code of the service.

6.3 Further Research

With the study's implementation, GVM does not notify the user when a new vulnerability report is available, so it must be periodically checked. The tool provides an alert system with several methods for notifying, such as Email and HTTP Get, among others. A practical use case for an alert would be when GVM detects a vulnerability with a CVSS over 4.0, which is considered a medium-level vulnerability.

Some minimal changes may be insignificant to the security of services, and it could be helpful to be able to inform the system that this service does not require testing at this time. On the other hand, requesting immediate security testing could be

handled with similar actions. These actions could be handled within the CI pipeline with, for example, using specified tags in the pull request's commit message.

To provide an answer for the reliability issue, another vulnerability scanner tool could be implemented to be run with GVM. A possible candidate would be OWASP ZAP that was already mentioned in the comparison of tools. ZAP also provides an API for automation, and it may be possible to integrate with GVM as an external scanner software.

References

Abildskov, J. 2020. What is DevOps?. Blog post on Eficode webpage. Accessed on 7 May 2021. Retrieved from https://www.eficode.com/blog/what-is-devops.

About CircleCI. N.d. CircleCI docs-webpage. Accessed on 1 May 2021. Retrieved from https://circleci.com/docs/2.0/about-circleci/?section=getting-started.

About Google Cloud services. N.d. Google Cloud docs-webpage. Accessed on 1 May 2021. Retrieved from https://cloud.google.com/docs/overview/cloud-platform-services.

Ansari, J. & Najera-Gutierrez, G. 2018. Web Penetration Testing with Kali Linux. 3rd ed. Birmingham: Packt Publishing Ltd.

Asay, M. 2017. It's a decade since DevOps became a 'thing' – and people still don't know what it means. Article on The Register webpage. Accessed on 7 May 2021. Retrieved from https://www.theregister.com/2017/12/08/devops_real_talk/.

Containers vs. virtual machines. 2019. Microsoft docs-webpage. Accessed on 16 May 2021. Retrieved from https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm.

Develop with Docker Engine API. N.d. Docker docs-webpage. Accessed on 22 March 2021. Retrieved from https://docs.docker.com/engine/api/.

Docker overview. N.d. Docker docs-webpage. Accessed on 22 March 2021. Retrieved from https://docs.docker.com/get-started/overview/.

Dockerfile reference. N.d. Docker docs-webpage. Accessed on 1 May 2021. Retrieved from https://docs.docker.com/engine/reference/builder/.

Duvall, P. N.d. Continuous Integration: Patterns and Anti-Patterns. PDF Document. Retrieved from http://www.cheat-sheets.org/saved-copy/rc084-010d-continuous-integration_1.pdf.

General Python FAQ. N.d. Python docs-webpage. Accessed on 12 May 2021. Retrieved from https://docs.python.org/3/faq/general.html.

Government, in cooperation with the President of the Republic, declares a state of emergency in Finland over coronavirus outbreak. 2020. Finnish government press release 140/2020 on 16 March 2020. Accessed on 21 January 2021. Retrieved from https://valtioneuvosto.fi/en/-/10616/hallitus-totesi-suomen-olevan-poikkeusoloissa-koronavirustilanteen-vuoksi.

Greenbone Management Protocol. N.d. Greenbone docs-webpage. Accessed on 2 May 2021. Retrieved from https://docs.greenbone.net/API/GMP/gmp-9.0.html.

Gvm-tools. N.d. Gvm-tools GitHub-page. Accessed on 11 May 2021. Retrieved from https://github.com/greenbone/gvm-tools.

ICalendar. N.d. ICalendar webpage. Accessed on 11 May 2021. Retrieved from https://icalendar.org.

Internet organised crime threat assessment IOCTA 2020. 2020. PDF Document. Europol. Retrieved from https://www.europol.europa.eu/activities-services/main-reports/internet-organised-crime-threat-assessment-iocta-2020.

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä: Kehittämistutkimuksen kirjoittamisen käytännön opas [Action research as a thesis: a practical guide to writing action research]. Jyväskylä: Jyväskylän ammattikorkeakoulun julkaisuja.

Kananen, J. 2015. Opinnäytetyön kirjoittajan opas: näin kirjoitan opinnäytetyön tai pro gradun alusta loppuun [Thesis writer's guide: this is how I write a thesis or a master's thesis from start to finish]. Jyväskylä: Jyväskylän ammattikorkeakoulu, Liiketoimintayksikkö.

Liveto. N.d. Liveto-webpage. Accessed on 10 January 2021. Retrieved from https://www.liveto.io/.

Liveto Group Oy. N.d. Finder-webpage. Accessed on 22 January 2021. Retrieved from https://www.finder.fi/Tapahtumat/Liveto+Group+Oy/Jyv%C3%A4skyl%C3%A4/yhteystiedot/3045895.

Lukka, K. 2001. Konstruktiivinen tutkimusote [Constructive research approach]. Metodix-webpage. Accessed on 5 May 2021. Retrieved from https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/.

Matteson, S. 2017. DevSecOps: What it is and how it can help you innovate in cybersecurity. Article on ZDNet-webpage. Accessed on 9 May 2021. Retrieved from https://www.zdnet.com/article/devsecops-what-it-is-and-how-it-can-help-you-innovate-in-cybersecurity/.

Mueller, E. 2010 (Revised 2019). What Is DevOps?. Blog post on The Agile Admin webpage. Accessed on 9 May 2021. Retrieved from https://theagileadmin.com/what-is-devops/.

Nessus professional. N.d. Tenable-webpage. Accessed on 5 May 2021. Retrieved from https://www.tenable.com/products/nessus/nessus-professional.

Ojasalo, K., Moilanen, T. & Ritalahti, J. 2015. Kehittämistyön menetelmät: Uudenlaista osaamista liiketoimintaan [Methods of development: New kind of expertise in business]. 3rd-4th ed. Helsinki: Sanoma Pro Oy.

OpenVAS. N.d. OpenVAS webpage. Accessed on 30 April 2021. Retrieved from https://www.openvas.org/.

Overview of Docker Compose. N.d. Docker docs-webpage. Accessed on 1 May 2021. Retrieved from https://docs.docker.com/compose/.

OWASP Zed Attack Proxy (ZAP). N.d. ZAP's webpage. Accessed on 10 May 2021. Retrieved from https://www.zaproxy.org/.

Razavi, M. 2021. GVM-containers. GitHub-repository. Accessed on 6 May 2021. Retrieved from https://github.com/admirito/gvm-containers.

Schenker, G. 2018. Learn Docker – Fundamentals of Docker 18. x: Everything You Need to Know about Containerizing Your Applications and Running Them in Production. Packt Publishing.

Seitz, J. 2014. Black Hat Python: Python Programming for Hackers and Pentesters. San Francisco: No Starch Press, Inc.

Sharma, S. 2017. The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise. John Wiley & Sons, Inc.

Viitasuo, E. 2020. Adding security testing in DevOps software development with continuous integration and continuous delivery practices. Jyväskylä: JAMK University of Applied Sciences. Bachelor's thesis. Retrieved from http://urn.fi/URN:NBN:fi:amk-2020060316773.

Wagner, J. 2019. About Greenbone Community Feed (GCF). Greenbone community forum post. Accessed on 1 May 2021. Retrieved from https://community.greenbone.net/t/about-greenbone-community-feed-gcf/1224

What is a Container?. N.d. Docker's webpage. Accessed on 22 March 2021. Retrieved from https://www.docker.com/resources/what-container.

What Is A Reverse Proxy?. N.d. Cloudflare's webpage. Accessed on 1 May 2021. Retrieved from https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/.

What is Docker?. N.d. Opensource.com-webpage. Accessed on 22 March 2021. Retrieved from https://opensource.com/resources/what-docker.

What is Nginx?. N.d. Nginx's webpage. Accessed on 1 May 2021. Retrieved from https://www.nginx.com/resources/glossary/nginx/.

What's New In Python 3.8. N.d. Python docs-webpage. Accessed on 16 May 2021. Retrieved from https://docs.python.org/3/whatsnew/3.8.html.

Why Docker?. N.d. Docker's webpage. Accessed on 22 March 2021. Retrieved from https://www.docker.com/why-docker.

Appendices

Appendix 1. GVM-containers' docker-compose.yml file.

```
version: '2.1'

volumes:
  redis-data: {}
  openvas-var-lib: {}
  gvm-var-lib: {}
  postgres-data: {}
  run-redis: {}
  run-ospd: {}
  logs: {}


services:
  gvm-postgres:
    image: admirito/gvm-postgres:11
    environment:
      PGDATA: /var/lib/postgresql/data
      POSTGRES_DB: gvmd
      POSTGRES_PASSWORD: password
      POSTGRES_USER: gvmduser
    volumes:
      - postgres-data:/var/lib/postgresql/data
    restart: always


  gvmd:
    # CONNECTED /var/run/ospd/ospd.sock
    build: ./gvmd
    environment:
      GVMD_POSTGRESQL_URI: postgresql://gvmduser:password@gvm-
postgres:5432/gvmd?application_name=gvmd
      GVMD_USER: admin
    volumes:
      - openvas-var-lib:/var/lib/openvas
      - gvm-var-lib:/var/lib/gvm
      - run-redis:/var/run/redis
      - run-ospd:/var/run/ospd
      - logs/:/var/log/gvm/
    mem_limit: 1000m
    depends_on:
      gvm-postgres:
        condition: service_started
    restart: always


  gsad:
    build: ./gsad
    environment:
      GVMD_HOST: gvmd
      GVMD_PORT: '9390'
    volumes:
      - logs/:/var/log/gvm/
    depends_on:
```

```
54      gvmd:
55        condition: service_started
56
57    restart: always
58
59
60  openvas:
61    # LISTENING /var/run/ospd/ospd.sock
62    # CONNECTED /var/run/redis/redis.sock
63    build: ./openvas
64    privileged: true
65    sysctls:
66      net.core.somaxconn: '2048'
67    volumes:
68      - openvas-var-lib:/var/lib/openvas
69      - run-redis:/var/run/redis
70      - run-ospd:/var/run/ospd
71      - logs/:/var/log/gvm/
72    depends_on:
73      gvm-postgres:
74        condition: service_started
75    restart: always
76
77
78  redis:
79    # LISTENING /var/run/redis/redis.sock
80    image: redis:5.0
81    volumes:
82    - run-redis:/var/run/redis
83    - redis-data:/data
84    command: redis-server—port 0 --unixsocket
85 /var/run/redis/redis.sock—unixsocketperm 755
86    privileged: true
87    sysctls:
88      net.core.somaxconn: '2048'
89    depends_on:
90      openvas:
91        condition: service_started
92    restart: always
```

Appendix 2. Python program for GMP - gmp.py.

```
import os
import sys

import arghandler

from gvm.connections import TLSConnection
from gvm.protocols.gmp import Gmp
from gvm.transforms import EtreeTransform
from gvm.xml import pretty_print


try:
    GVM_HOST = os.environ['GVM_HOST']
    GVM_USER = os.environ['GVM_USER']
    GVM_PASS = os.environ['GVM_PASS']
except:
    GVM_USER = 'gvm'
    GVM_HOST = 'gvmthesis.tk'
    GVM_PASS = 'notthepassword'


def create_target(gmp, args=None):
    if not (name := args.name):
        name = args.host

    return gmp.create_target(
        name=name,
        hosts=[args.host],
        comment=args.comment
    )


def get_config_id(config_index: int) -> str:
    requested_config = arghandler.SCAN_CONFIGS[config_index]

    config_resp = gmp.get_configs()
    configs = zip(
        config_resp.xpath('config/name/text()'),
        config_resp.xpath('config/@id')
    )

    for name, config_id in configs:
        if name == requested_config:
            return config_id


def create_task(gmp, args=None):
    if not (name := args.name):
        name = args.target  # fallback to target
    if schedule := args.schedule:
        schedule = get_schedule_by_name(gmp, args=args)
    target = get_target_by_name(gmp, name)

    create_task_resp = gmp.create_task(
        name=name,
        config_id=get_config_id(args.scan_config),
        target_id=target,
```

```
58          scanner_id=gmp.get_scanners().xpath('scanner[2]/@id')[0],
59          schedule_id=schedule,
60          comment=args.comment
61      )
62
63      if args.start_task:  # with -S
64          start_task(gmp, args=args)
65
66      return create_task_resp
67
68
69  def get_schedule_by_name(gmp, args=None):
70      schedules = get_schedules(gmp)
71
72      if not (name := args.name):
73          name = args.target
74
75      requested_schedule = '{} {}'.format(name, args.schedule)
76      for schedule_name, schedule_id in schedules:
77          if schedule_name == requested_schedule:
78              return schedule_id
79
80      create_schedule_resp = create_schedule(gmp, args=args)
81      return get_schedule_by_name(gmp, args=args)
82
83
84  def get_schedules(gmp) -> zip:
85      schedule_resp = gmp.get_schedules()
86      return zip(
87          schedule_resp.xpath('schedule/name/text()'),
88          schedule_resp.xpath('schedule/@id')
89      )
90
91
92  def create_schedule(gmp, args=None):
93      import pytz
94      from datetime import date, datetime, timedelta
95      from icalendar import Calendar, Event
96
97      cal = Calendar()
98      cal.add('prodid', '1')
99      cal.add('version', '2.0')
100     event = Event()
101     tomorrow = date.today() + timedelta(days=1)
102     event.add('dtstamp',
103               datetime.now(tz=pytz.timezone('Europe/Helsinki')))
104     event.add('dtstart',
105               datetime.combine(tomorrow,
106                        tzinfo=pytz.timezone('Europe/Helsinki')))
107
108     if freq := args.schedule:
109         event.add('rrule', {'freq': freq})
110
111     cal.add_component(event)
112
113     if not (name := args.name):
114         name = args.target
115
116     return gmp.create_schedule(
```

```
117         name='{} {}'.format(name, freq),
118         timezone='Europe/Helsinki',
119         icalendar=cal.to_ical()
120     )
121
122
123 def get_target_by_name(gmp, requested_target):
124     targets = get_targets(gmp)
125     for target_name, target_id in targets:
126         if target_name == requested_target:
127             return target_id
128
129
130 def get_targets(gmp) -> zip():
131     targets_resp = gmp.get_targets()
132     return zip(
133         targets_resp.xpath('target/name/text()'),
134         targets_resp.xpath('target/@id')
135     )
136
137
138 def list_targets(gmp, **kwargs):
139     targets = get_targets(gmp)
140     print(list(targets))
141
142
143 def get_task_by_name(gmp, **kwargs) -> str:
144     if args.start_task is True:
145         requested_task = args.name
146     else:
147         requested_task = args.start_task
148
149     tasks = get_tasks(gmp)
150     for task_name, task_id in tasks:
151         if task_name == requested_task:
152             return task_id
153
154
155 def get_tasks(gmp, **kwargs) -> zip():
156     tasks_resp = gmp.get_tasks()
157     return zip(
158         tasks_resp.xpath('task/name/text()'),
159         tasks_resp.xpath('task/@id')
160     )
161
162
163 def list_tasks(gmp, **kwargs):
164     tasks = get_tasks(gmp, **kwargs)
165     print(list(tasks))
166
167
168 def start_task(gmp, **kwargs):
169     task_id = get_task_by_name(gmp, **kwargs)
170     start_task_resp = gmp.start_task(task_id)
171     return start_task_resp
172
173
174 def get_function(args):
175     if args.add_target:
```

```
176         return create_target
177     elif args.add_task:
178         return create_task
179     elif args.list_targets:
180         return list_targets
181     elif args.list_tasks:
182         return list_tasks
183     elif args.start_task:
184         return start_task
185 
186 
187 def login(gmp) -> bool:
188     try:
189         auth = gmp.authenticate(GVM_USER, GVM_PASS)
190         status = auth.xpath('@status')
191 
192         if '400' in status:
193             print('login failed')
194             raise Exception
195         print('login successful\n')
196         return True
197 
198     except Exception as e:
199         print(e)
200 
201 
202 if __name__ == '__main__':
203     args = arghandler.handle()
204     print(args)
205 
206     if len(sys.argv) > 1:  # require >= 1 arguments
207         function = get_function(args)
208 
209         connection = TLSConnection(hostname=GVM_HOST)  # 9390
210         transform = EtreeTransform()
211 
212         with Gmp(connection, transform=transform) as gmp:
213             if login(gmp):
214                 print(function.__name__)
215                 out = function(gmp, args=args)
216                 pretty_print(out)
217     else:
218         print('Arguments required')
219 
```

Appendix 3. Python argument parser for GMP – arghandler.py.

```
import argparse


SCAN_CONFIGS = {
    0: 'Base',
    1: 'Discovery',
    2: 'Full and fast',
    3: 'Full and fast ultimate',
    4: 'Full and very deep',
    5: 'Full and very deep ultimate',
    6: 'Host Discovery',
    7: 'System Discovery',
}

def handle():
    parser = argparse.ArgumentParser()
    exclusive = parser.add_mutually_exclusive_group()

    exclusive.add_argument(
        '-a', '--add-target',
        action='store_true',
        help='Add a new target'
    )
    exclusive.add_argument(
        '-A', '--add-task',
        action='store_true',
        help='Add a new task'
    )

    # Required Flags
    parser.add_argument(
        '-C', '--scan-config',
        type=int,
        choices=range(0, 9),
        help=str(SCAN_CONFIGS)
    )
    parser.add_argument(
        '-t', '--target',
        type=str,
        help='ID or name of the target, required with add-task'
    )
    parser.add_argument(
        '-H', '--host',
        type=str,
        help='IP address or hostname for new target'
    )

    # Non-required Flags
    parser.add_argument(
        '-n', '--name',
        type=str,
        help='Name for target/task, required with add-target and
add-task'
    )
    parser.add_argument(
        '-c', '--comment',
        type=str,
```

```
58          help='Add comment for created object'
59      )
60      parser.add_argument(
61          '-S', '--start',
62          action='store_true',
63          help='Start task'
64      )
65      parser.add_argument(
66          '-s', '--schedule',
67          type=str,
68          choices=['now', 'daily', 'weekly'],
69          help='Set task schedule'
70      )
71
72      # List flags
73      parser.add_argument(
74          '-lt', '--list-targets',
75          action='store_true',
76          help='List target names and IDs'
77      )
78      parser.add_argument(
79          '-lT', '--list-tasks',
80          action='store_true',
81          help='List task names and IDs'
82      )
83
84      args = parser.parse_args()
85
86      # Check for missing arguments
87      if args.add_task:
88          assert (args.target is not None
89                  and args.scan_config is not None),\
90              parser.error(
91                  'MISSING REQUIRED ARGUMENT\n \
92                  -t|--target <NAME|ID> \n \
93                  -C|--scan-config <0-8> \n \
94                  WHEN -A|--add-task is set'
95              )
96
97      elif args.add_target:
98          assert args.host is not None,\
99              parser.error(
100                 'MISSING REQUIRED ARGUMENT\n \
101                 -h|--host <IP|HOSTNAME> \n \
102                 WHEN \n \
103                 -a|--add-target is set'
104             )
105
106     return args
```