UCLouvain

École polytechnique de Louvain

Clustering hybrid cyber ranges for cybersecurity education purposes

Author: **Jonathan DE SALLE**
Supervisors: **Axel LEGAY, Benoît DUHOUX**
Reader: **Cristel PELSSER**
Academic year 2022–2023
Master [120] in Computer Science

Contents

Abstract

This Master's thesis objective is to develop a cyber range infrastructure based on a cluster of small computers, in a way that would allow us to answer the following question: "How to build a cyber range infrastructure to run training scenarios while preserving low cost and low energy consumption?" It will also be important to assess this cyber range implementation based on its functionality, costs, and suitability for cybersecurity education.

The main contributions of this Master's thesis consist on the development of a new framework to deploy "hybrid cyber ranges". The hybrid characteristics of our cyber range implementation allow us to deploy ARM-based workload as well as x86-64 based workload, and allows us to easily deploy both VM and containers depending of our needs. It allows the development of new scenarios, while maintaining compatibility with general scripts allowing both ease of use, reusability of components and modularity. The physical implementation of this cyber range was built over a cluster of Intel NUC and raspberry Pi and have an initial cost of less than 700 euros to replicate. During our test, The maximum power used by our cyber range was 35 watts during our stress test, for a general consumption around 20 watts. All in one, our cyber range seems to be an affordable solution for educative institutions.

Acknowledgements

I would like to take this page to thank all the people who made this work possible.

First of all, I would like to express my sincere gratitude to the member of my jury, the supervisors and readers of this Master's thesis:

- **Pr. Axel Legay** for not only letting me work on a subject I proposed and believing in my initial ideas, but also for his dedicated supervision of this Master's thesis and the time dedicated to this purpose.
- **Dr. Benoît Duhoux** for his continuous advices and remarks that always pushed me to improve on my ideas and draft, and for his constant positive pressure that helped me to maintain my best motivation all along.
- **Pr. Cristel Pelsser**, for her contribution as a reader to this Master's thesis and for taking time out of her busy schedule to be part of my jury.

Then I would like to thank all the friends that supported and encouraged me during the period of writing this Master's thesis. Especially to those who helped me more directly, by reading my drafts to help notice the parts where I wasn't clear enough. A special mention to **Emile Villette** and **Samy Bettaieb**.

Finally, I would like to thank my family for their continuous support during my whole university curriculum and, well, my life in general. Special shout out to my parents, **Hervé de Salle** and **Nathalie Vanlanguenakers** for having always been there for me, whenever I needed them (and also when I didn't...) ♡.

Chapter 1

Introduction

In the context of an increasingly digital world, where cyber threats are becoming more prevalent and more sophisticated, the need for cybersecurity experts is continuously growing. Cybersecurity education, on the other hand, is a very practical subject, theory alone is not able to develop cyber security experts. Cyber ranges are a way to give students such a hand-on training to develop their skills.

Unfortunately, cyber ranges are quite expensive, and while the cost may be bearable by companies it can still be often prohibitive to educative initiatives. While several solutions have been proposed, the 2020's paper of Oh *et al.*[1] describing a cyber range infrastructure based on containers and a cluster of Raspberry Pi shows an interesting direction to investigate for designing education-geared cyber ranges infrastructure. They did, however, note a lack of ARM-based security-oriented containers that limited them in this endeavor.

This Master's thesis objective is to develop a cyber range infrastructure based on a cluster of small computers, in a way that would allow us to answer the following question: "How to build a cyber range infrastructure to run training scenarios while preserving low cost and low energy consumption?" It will also be important to assess this cyber range implementation based on its functionality, costs, and suitability for cybersecurity education.

The main contributions of this Master's thesis consist on the development of a new framework to deploy "hybrid cyber ranges". The hybrid characteristics of our cyber range implementation allow us to deploy ARM-based workload as well as x86-64 based workload, and allows us to easily deploy both VM and containers depending of our needs. It allows the development of new scenarios, while maintaining compatibility with general scripts allowing both ease of use, reusability of components and modularity. The actual implementation of this cyber range was

made with a cluster of Intel NUC and raspberry Pi and have an initial cost of less than 700 euros to replicate. During our test, The maximu power used by our cyber range was 35 watts during our stress test, for a general consumption around 20 watts. All in one, our cyber range seems to be an affordable solution for educative institutions.

In the following chapters of this Master's thesis, we will start by describing the state of the art in two chapters: the second chapter, "Background material" will focus on concepts and technologies to the framework described in this master thesis, while the third chapter, "Related works" will focus on a non-exhaustive overview of research landscape of cyber range development. After that, the next two chapter focus on our implementation: the fourth chapter will explain our approach, while the fifth chapter will explain the case studies we developed to test our infrastructure. We will then assess our infrastructure in chapter seven, "Validation". Finally, we will explain some idea about future possibilities of development for our current implementation by focusing on some of its limits in chapter 8, "Future Work", before concluding this work.

Chapter 2

Background material

This chapter aims to define key principles and concepts used in the scope of this master thesis. We will first define cyber ranges before quickly introducing CPU architectures, virtualization technologies and containers and how they are implemented in Kubernetes and Kubevirt, which are used in our solution.

2.1 Cyber Ranges

Several common terms must be defined to ensure a correct grasp of the terminology used in the study of cyber ranges.

Cyber ranges are infrastructure built for the explicit purpose of training in cybersecurity. It is generally implemented in the form of a virtual environment simulating a network (in its entirety or only part of it) to allow the trainee to explore and to learn about vulnerable applications, exploitation techniques and/or their mitigation[1].The different users of a cyber range and their interaction are illustrated *fig 2.1*.

A training exercise or scenario is a training instance deployed on the cyber range. It is defined by its context (and related infrastructure), its objectives and the steps needed to go from the initial state to the completion of the objectives. A given cyber range can be used to hold different exercises either simultaneously or at different times.

The context of a scenario includes its initial state, the information known by each user and the different components of the simulated infrastructure. The initial context of an exercise given to the trainee can either contain all possible information (white box training), partial information (grey box training) or no information other than limitations and an initial IP range (black box training).

The cyber range designer is the person who develops a cyber range infrastructure. The scenario designer is the person who creates a scenario for training purposes.

A trainer is the person managing the cyber range to deploy specific scenarios to teach students about concepts of cybersecurity.

The trainees are the students receiving education in cybersecurity through the use of the cyber range by participating in a scenario.

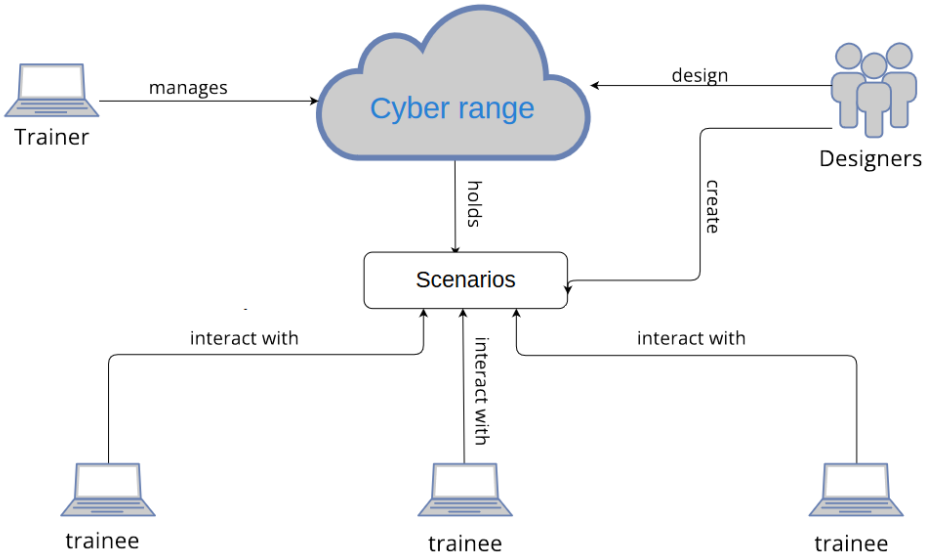

Figure 2.1: Illustration of the interactions

2.2 CPU Architecture

The term CPU architecture refers to the instruction set used by the processor, a program compiled for a specific instruction set won't work on a processor of another architecture. We will focus on the x86-64 and the ARM architectures. While a description of the differences between the x86-64 and the ARM architectures falls out of the scope of this Master's thesis, a general guideline would be to say that usually desktop and laptop computer are made under the x86-64 architecture while smaller devices such as smartphones, tablets, *etc* follow the ARM architecture.

2.3 Virtualization

Virtualization is a concept allowing the abstraction of resources from the actual physical resources at disposition.A single physical resource can thus serve as multiple virtual resources independent from each other [2] . This abstraction allows us to deploy a complete infrastructure on a limited number of physical machines.

Depending on the level of abstraction, isolation and technologies used, we can define different types of virtualization. In the scope of this Master's thesis, we will focus on Hypervisor based virtual machines and containers which we will describe in the following subsection before explaining the main difference between containers and virtual machine. The different types of virtualization described in this Master's thesis and their comparison to real machines are illustrated in *fig 2.2*.

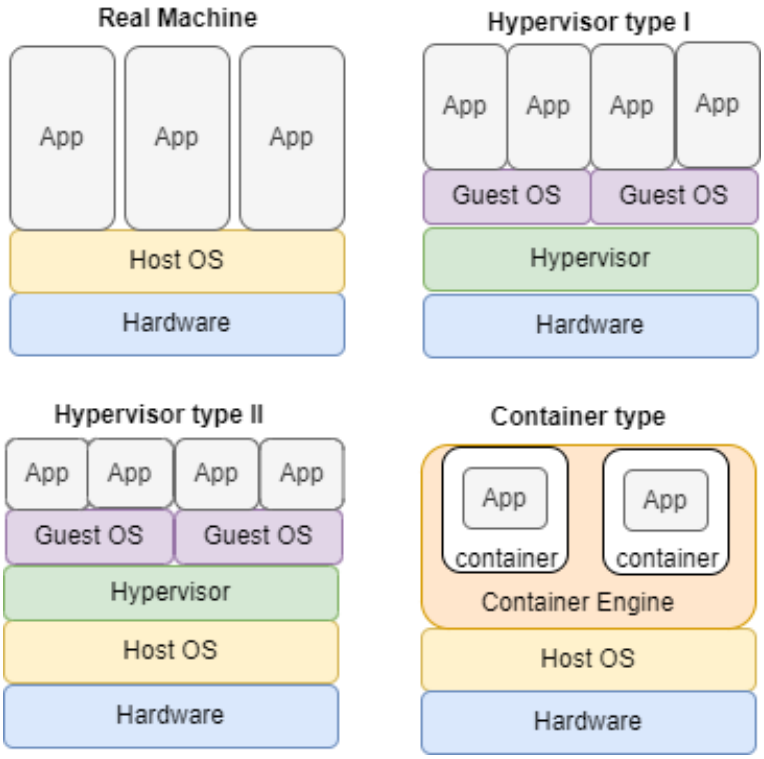

Figure 2.2: overview of virtualization types [3]

2.3.1 Hypervisor based virtual machines

A Hypervisor (also called "Virtual Machine Monitor") is a *"software layer that virtualizes all of the resources of a physical machine"* [4]. Hypervisors can be

separated into two types. Type I hypervisor run directly on the hardware of the physical machine and do not necessitate another OS to run on the host. They are best used on machines that are only used in the deployment on virtual machines. Examples of such hypervisors would include Microsoft Hyper-v and VMWare ESX.

Type II hypervisors run as a software that runs over another operating system, referenced as the "host operating system", over which will run a "guest OS". Examples of such hypervisors would include Virtualbox, VMWare workstation or QEMU.

Using a hypervisor, either of type I or of type 2, allows running virtual machines with unmodified OS.

2.3.2 Containers

Containers are virtual environments that encapsulate complete applications and their dependencies into a single component, the container itself, that offers essential features such as isolation and resource management within a Linux environment [5].

Containers achieve these properties through the utilization of three core mechanisms and policies provided by Linux: chroot, cgroups, and namespaces.

Firstly, Chroot [6] allows a process to be run with a different root directory, which allows the container to access a separate file system, distinct from other processes.

In addition, namespaces[7] are a mechanism that allows isolation between processes. Processes within a same namespace seem to have their own instances of global resources and change on these resources are only visible to processes in that same namespace.

Finally, Cgroups [8] serves as a policy control mechanism over sharing resources.

Containers have been particularly popularized by the rise of Docker [9], an Open-Source platform helping users to build and manage containers as self-contained images.

2.3.3 Containers vs Virtual machines

While a virtual machine virtualizes entire machine and operating system,container operate on a software level and only virtualize application running over an operating system, while still relying on the underlying operating system for kernel actions. While virtual machines allow better isolation with the host operating system, containers are more efficient and use less computing ressources. While both have their place in a development environment, the energy efficiency of containers allows

us a more efficient use of resources for the deployment of a cyber range, in addition to provide us with a better flexibility in modifying components. .

A concern that may arise when thinking about the idea of using containers in a cyber range is the issue of vulnerability reproducibility. However a 2020 appear by Nakata *et al.*[10] shows that containers allow 99.3% of vulnerability reproducibility compared to an environment deployed around virtual machines.

2.4 Clustering

Clustering allows different nodes to work together on a workload. A cluster is made of a control plane and workers nodes. *fig 2.3* illustrates the components of a Kubernetes cluster holding both containers and virtual machines, and the interactions between those components.

This section will aim to describe those components and the technology behind them. The first subsection will describe Kubernetes and its components, including the control plane. The following subsection will describe Kubervirt and its interaction with Kubernetes to handle virtual machines.

2.4.1 Kubernetes

Kubernetes is an open source container orchestration engine, which allows the automated deployment, scaling, and management of containerized applications[11]. It allows to create and distribute pods and services among the different nodes of an established cluster. A pod is the minimal unit that can be processed in Kubernetes and correspond to a group of containers sharing the same namespace network and volume that are deployed on a same physical node.

To explains the architecture of our infrastructure in later chapters, it is important to understand the basic architecture of Kubernetes cluster architecture . We will summarize those basics in 3 parts: Kubernetes object, the different types of nodes in the cluster and their components and how the networking is handled in Kubernetes. Since those subjects are quite complex and a complete explanation of those could be a work on its own, we will focus on the aspects directly influencing the conception of our infrastructure. This section is inspired by the Kubernetes documentation. [11]

2.4.1.1 Objects

An object is a permanent entity in Kubernetes that represents a state of the cluster. Objects are declared in a YAML configuration file and constitute a declaration of

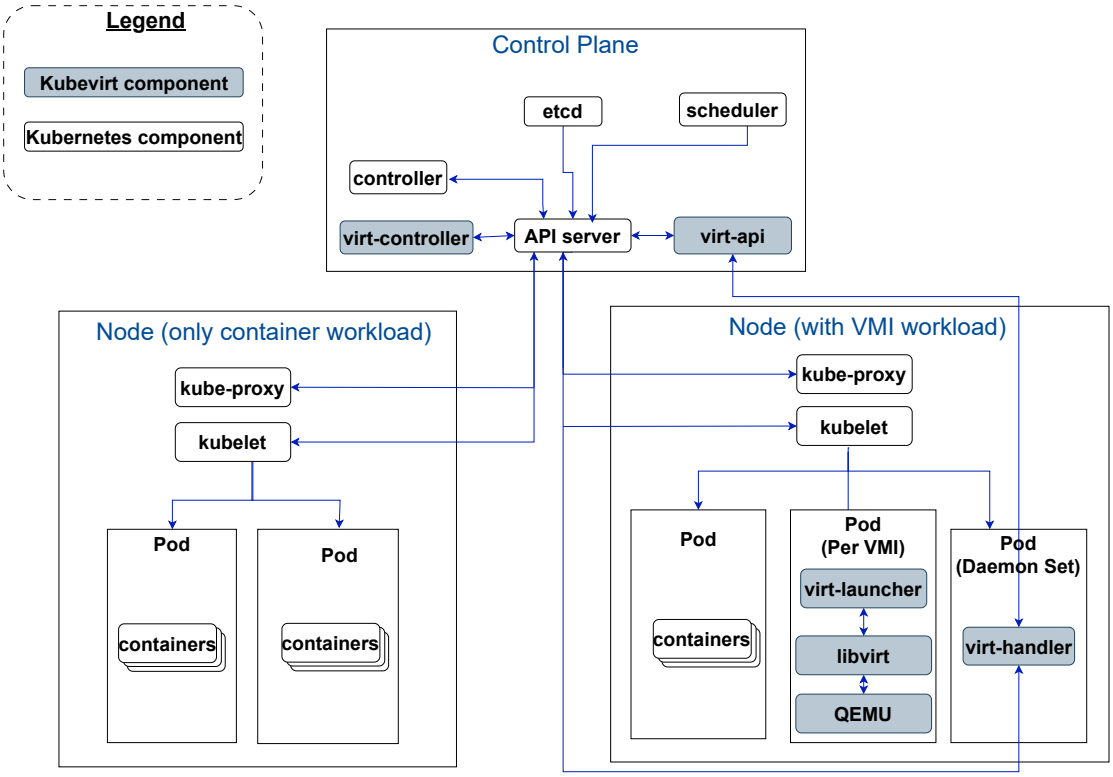

Figure 2.3: overview of the components of a Kubernetes/Kubevirt cluster and their interactions

intent, once an object is created in Kubernetes, the Kubernetes engine will always try to ensure the realization of its specifications.

2.4.1.2 Nodes

In a Kubernetes Cluster, there are 2 types of nodes: Control Nodes and Worker Nodes. While the control nodes manage the control plane, the worker nodes support the actual workload of our scenarios.

The diagram in *fig 2.4* illustrates the different components of a Kubernetes cluster and its two distinct parts.

The Control Plane hold at least four pods: the API server, the scheduler, the controller, and etcd. The API server pod serves as the front end of our control plane. It exposes the Kubernetes API from the control planes to all nodes in our cluster.

The scheduler pod is responsible for selecting nodes to deploy pods that are yet to be deployed on the cluster.

The controller pod hold the different controller processes. It is the pod actually controlling jobs, nodes, *etc.*

Finally, Etcd is a pod that store datas about the cluster's configuration.

In addition to the pods running the containers corresponding to the actual workload of the cluster, worker nodes require two essential pods to function within the cluster: kube-proxy and kubelet.

The kubelet pod ensures that the different pods runs on each node of the cluster and monitors the current status of the containers within these pods.

The Kube-proxy pod is a network proxy maintaining network rules on all nodes of our cluster. this allows communication between each node of our cluster and with external entities based on the service's configuration.

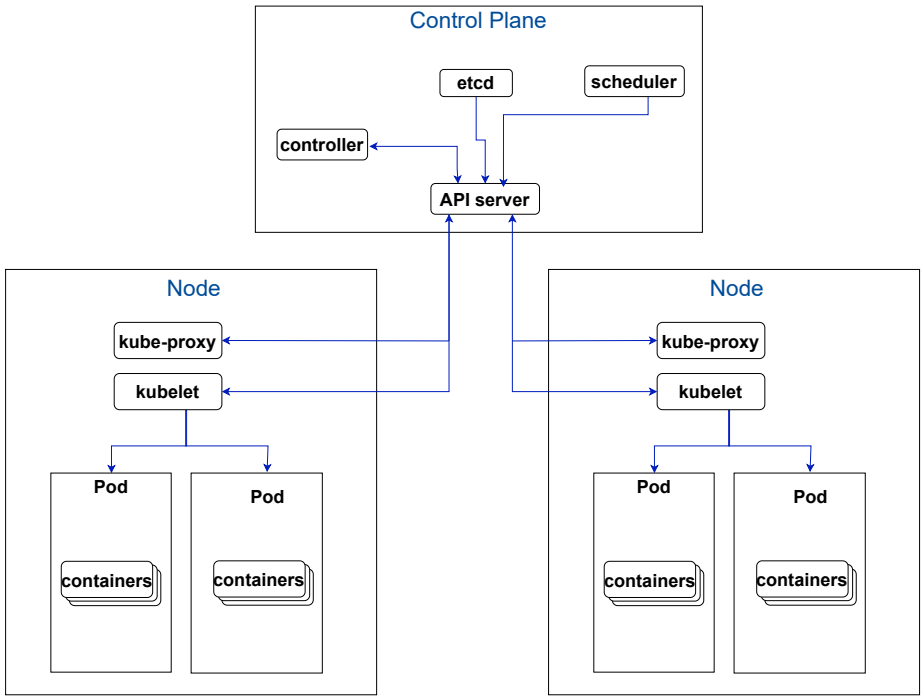

Figure 2.4: Interactions between nodes and control plane

2.4.1.3 Networking

Kubernets consider that the network communication between pods in the cluster is automatically managed and that each pod get a unique IP address for the namespace of the cluster, allowing communication between every pods. To create this overlay network, Kubernetes requires a type of plugin called a Container Network Interface (CNI). There are a few different CNI available to use with Kubernetes such as Flannel, Calico, Canal, *etc.*

Unfortunately, IP addresses provided by the CNI are not reachable from outside the cluster. To allow external usage, Kubernetes use services. Services allow to expose a Kubernetes application running on one or several pods using a single endpoint. A single set of <IP:ports> can therefore represent a complete application running on several pods.

There are different types of services , but we will focus on the LoadBalancer type. Loadbalancer type allow the exposure of a service to a single set of <IP:ports> that will be accessible from outside the cluster. While configurations of a this type are possible by using the IP of the nodes as a gateway, such a service usually requires an external Load Balancer service, such as the one provided by the cloud companies or by add-on such as metalLB.

2.4.2 Kubevirt

Kubevirt [12] is an add-on to Kubernetes allowing the deployment and management of virtual machines over a Kubernetes cluster and interfaces. To enable these new possibilities, Kubervirt introduce new resources, additional controllers and new daemons to the Kubernetes API. This allows the use of a new type on Kubernetes: Virtual Machine Instances (VMI).

Interactions between the Kubevirt components and Kubernetes are illustrated in the diagram on *fig 2.5*. Kubevirt uses four pods to enable interactions between Kubernetes and VMIs: virt-controller, virt-api, virt handler and virt-launcher.

The virt-controller pod is holding all the controller processes. It controls the deployment, termination, *etc* of VMI instances, and comunicate with the kubernetes API.

Virt-api is the pod exposing the Kubevirt API and communicate its needs with the kubernetes API. It can be thought of as the front end of the Kubevirt ecosystem.

Virt-handler has an analog role to kubelet in Kubernetes. It allows the VMIs to run on each node of the cluster and monitor current status VMIs.

Virt-launcher is the pods running the actual VM workload. It works over libvirt[13] which itself works over QEMU[14], a type II hypervisor.

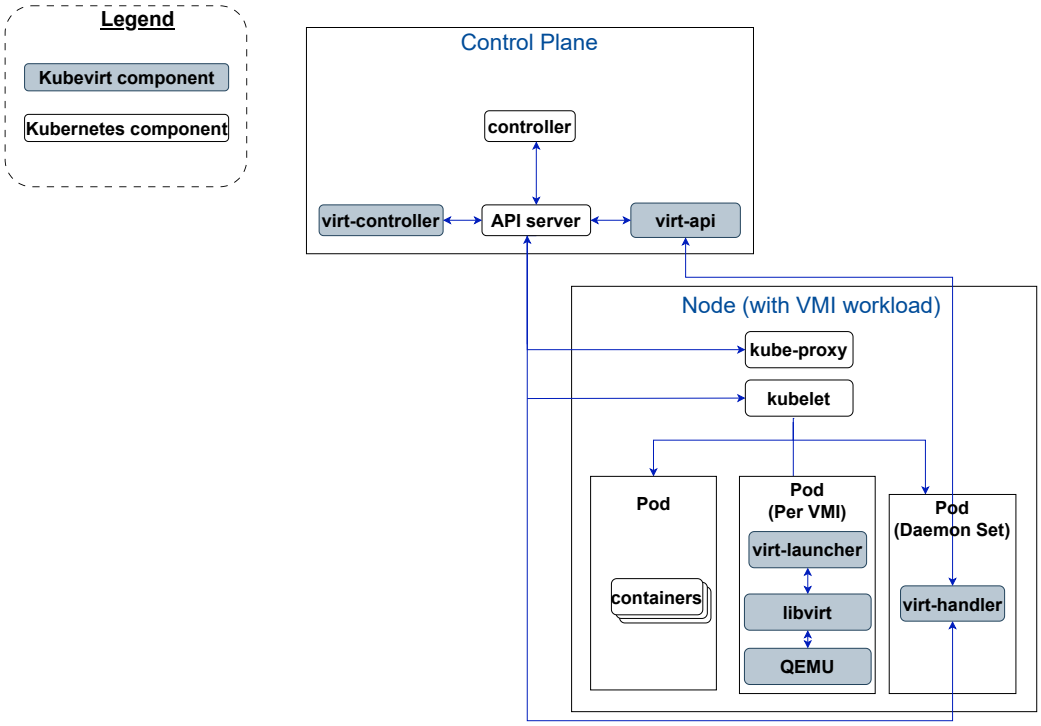

Figure 2.5: Kubevirt components inclusion to Kubernetes Architecture [12]

Chapter 3

Related Work

This chapter aims to give a quick introduction to the current state of the art of the field of cyber range design and implementation and to describe approaches currently taken in this endeavor. We will initially focus on infrastructures base on virtual machines, before talking about about container based cyber range implementations. Finally, we will introduce works related to a subject not developed in the framework proposed in this master thesis: scenario generation and randomization.

3.1 Virtual machine based implementations

Virtual machines are the most common virtualization technique used to deploy cyber range [3]. In a survey by Chouliaras *et al.*in 2021[15], among 10 cyber ranges studied in more detail, only one was using docker container instead , while all the others seemed to be deployed on hypervisor based platform or directly on VM deployed on IaaS platforms. A similar observation can be made from the survey made in 2020 by Yamin *et al.*, which analysed 100 articles about cyber ranges and test beds implementations.

In their 2017 paper [16], Beuran *et al.*present CyTrONE (Cybersecurity Training and Operation Netwok Environment), an open source framework automating both the scenario content generation and the set up of the environment. The training description and deployment procedure are configured thought text files formatted in YAML, and are instantiated through CyRIS[17]. CyRis is an automated instantiating framework for cyber range that instanciate libvirt based virtual machines over a network managed by OpenStack and VMware vSphere.

In 2019, the Belgian Royal military academy published a paper [18] about their cyber range infrastructure and insisting on the important of cyber range to develop

cybersecurity Awareness. In their implementation, scenario are described in a text format, eitherYAML or JSON and use Vagrant images to deploy their virtual machines. Their implementation is based on the type II hypervisor virtualbox, which itself relies on QEMU and KVM. All modification over the initial Vagrant image (virtual hardware, Operating system configuration, etc) are described in the yaml descriptions, which are can support different types of scenario.

3.2 Container based implementations

Containers base cyber range implementation are still a relatively new area of research. A first concern that researcher had about container based implementation was vulnerability reproducibility in such an infrastructure. In their 2020 paper [10], Nakata *et al.*feared that since container based and hypervisor based virtualization, which was used in cyber range implementation in previous works, are different, *"Individual vulnerabilities and incidents cannot be reproduced correctly and may not work as envisioned"*. They made a complete evaluation of vulnerability reproducibility in containers and virtual machines compared to a real environment and arrived at the conclusion that container had a 99.3% of vulnerability reproducibility and 1/10 of the ressources consumption of virtual machines.

In 2021, Nakata *et al.*thus published a paper on CyExec, a container-based cyber range allowing scenario randomization. CyExec uses Docker to create its scenario. Default components are described in a Docker compose descripton before the randomization process.

In 2020, Oh *et al.*proposed a cyber range infrastructure using docker containers over a cluster of Raspberry Pi. The objective of this paper was very similar to the objective of this thesis: developing an inexpensive cyber range infrastructure in the scope of cybersecurity education.In this paper, they deploy a DVWA on the Rapsberry Pi cluster using docker swarm. While this cyber range worked well, they faced limitation due to the ARM architecture of Raspberry Pi and the lack of cybersecurity container that are compatible wirth ARM. In march 2023, Legg *et al.*built upon this idea and developed a pi-lab [19], a portable cyber range used for educational activities in school. He also developed new cybersecurity themed container geared toward ARM devices for this activity.

3.3 Scenario generation and randomization

In this section, we will explore some work about scenario generation and randomization for cyber ranges.

In their CyExec paper [3], Nakata *et al.*describe the process of their scenario

randomization . CyExec randomization is based on the concepts of Milestones that have to be reached, for example "gaining root access". If multiples attack can be done to achieve this milestone, then the scenario can be randomized. To develop the vulnerabilities that allow to reach the milestones, Nakata *et al.*used the Metasploitable 2 exploit database and looked for exploit that would achieve these effects.

In 2019, Beuran *et al.*proposed a progression management system for CyTrONE [20].It makes use of "trigger" to allow an action to happens only when a certain condition is reached (per example, the end of a timer).Such an action may be giving the trainee a hint, or showing him a solution for a step for him not to be blocked at the same step for all the duration of the training.

Finally

Chapter 4

Approach

This chapter aims to explain the approach we have taken in order to design and implement our solution.

We will begin by explaining the motivations for this Master's thesis. We will then provide a high-level overview of our general approach and what led us to our choices, before going into more details about the materials used and how the cluster is implemented. After that, we will detail the process by which a designer can create a new scenario and the managing of the range by a trainer. Finally, we will discuss the process by which a trainee participates in the scenarios deployed on the cyber range.

4.1 Motivations

In the context of an increasingly digital world, cyber threats are becoming more and more sophisticated and the need for cybersecurity experts is continuously growing. However, cybersecurity training is a complex and practical subject. As Crumpler *et al.* explain in their 2019 paper, *"Theory alone does not prepare graduates for the tasks they will face once they step onto the job. Practical training and hands-on experience is necessary to equip students with the tangible skills employers expect"* [21] . While a practical education of cybersecurity is necessary, it is not possible to work on the production network of an enterprise for obvious reason. Nonetheless, simulating scenarios and incidents in a realistic way and learning in a realistic environment is the best way to create a good cybersecurity awareness and educate experts in cybersecurity [18].

A solution to create such a realistic environment is through the use of cyber ranges, which allow simulating entire infrastructure for the purpose of cybersecurity education. Unfortunately, the high cost associated with building and maintaining

cyber range infrastructure limit their access to smaller organization, including educative organization, since commercial cyber range solutions can be prohibitively expensive.

The purpose of this master thesis is thus to design a method to build inexpensive cyber range infrastructure for cybersecurity education. A first research question that guided us along this Master's thesis was "how to build a cyber range infrastructure to run training scenarios while preserving low cost and low energy consumption?". Once such a solution is designed, we have to make sure scenarios can be run on the infrastructure to teach students about specific concepts. We also have to make sure that our solution is not too expensive. That leads us to our second research question: "how to assess a cyber range infrastructure on its functionality, but also on its costs and energy efficiency? ".

While this chapter aims to answer the first research question, the following chapter, "Validation" aims to answer the second research question.

4.2 General approach

The cost of a cyber range is greatly impacted by the technologies used. The use of containers can lead to a very important diminution of the resources overhead and allows a good replication of environment and vulnerabilities[10].

The solution of using a Raspberry Pi cluster proposed by Oh *et al.*[1] shows a good direction to work on for an inexpensive education purposed cyber ranges. Unfortunately, the researchers found that solution limited by the underlying ARM architecture of the Raspberry Pi and the lack of ARM compatible security containers they observed. While the work of Legg *et al.*[19] try to fill this gap by creating more ARM based containers, we decided to use a different approach in order to not limit ourselves to the ARM architecture.

By using a cluster of Intel NUC, the problem of the ARM architecture seems to become nonexistent, since the NUC uses an x86_64 architecture. However, by itself, this solution would deprive us of the opportunity of working on ARM machines in our cyber range and would thus limit the possible learning experiences that could be given using our range. Indeed, ARM machines are becoming more and more prevalent, from the use of phones, tablets, or even the recent MacBook and their M1 puces, and some vulnerabilities only exist on ARM-based machines.

Such solution also limits us when working with some products. Some companies only ship their product in the form of a virtual machine, which means those

products could not be deployed in cyber range based only on containers.

Those different reasons led us to our approach, referred to as "hybrid cyber ranges". Our cyber range shows hybrid characteristics in two significant ways.

First, the Kubernetes based cluster that we create allows the integration of nodes based on both the ARM architecture and the x86-64 architecture , which allows us to use both architecture with their related containers with the limitation that each node can only handle a workload built for its specific architecture.
The second hybridization of our cluster comes from the inclusion of the Kubevirt add-on to Kubernetes. This add-on allows us not only to deploy the virtual machine based products on our cyber range but also to emulate an architecture for them, thanks to its QEMU inclusion.
The fusion of these elements, the mix of architectures and of containers and virtual machines, is what inspired us to call our approach "hybrid cyber ranges" to better illustrate the versatility of our design.

Another key aspect of our cyber range implementation is the use of multiple scripts to facilitate the deployment of both our cluster and the scenarios themselves. It allows our approach to be easier to use while staying general and modular: we separated the deployment of the scenario to the creation of the cluster itself, which could allow the use of different choices of Kubernetes deployment methods (kubeadm, minikube, *etc*), and allows us to deploy any scenario respecting our specifications on our cyber range.

4.3 Material

For the purpose of creating our cyber range, several pieces of hardware are needed: a router, some computers for the nodes of our cluster and a computer to manage the cluster. Since we want to create a hybrid range, we'll need to use two kinds of computer for the node: some of the nodes using a x86-64 architecture and others using an ARM architecture.

While the computer used to manage the range is irrelevant to the performances of the cyber range, we'll describe the router and the computers used in the range for reproducibility purposes.

4.3.1 Router

In our implementation we used a Mikrotik Hex Lite router, represented in *fig 4.1.* It runs over RouterOS [22], an operating system for routers running over the Linux Kernel developed by Mikrotik. In addition to being included on Mikrotik routers,

RouterOS can also be installed on a standard x86 computer with different levels of license.

The Mikrotik Hex Lite has a flash storage of 16 MB and a RAM size of 64 MB. The associated license for RouterOS is the level 4 license2 [1], which allow a lot more possibilities for configuration that what is needed for our implementation.

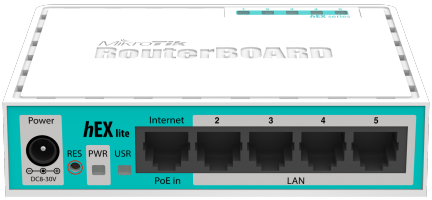

Figure 4.1: Mikrotik Hex Lite

4.3.2 x86-64 nodes

The X86-64 computer used in our cyber range is the Intel NUC model "NUC5PPYH", represented *fig 4.2*. Those NUC run over Ubuntu 22.04.2 LTS, which at the time of installation used a Linux 5.15 LTS kernel.

The NUCs of our infrastructure are equipped with an Intel Pentium N3700 CPU, which holds 4 cores and 2MB of cache. They are also installed with 8GB of DDR3 RAM.

Figure 4.2: Intel NUC model "NUC5PPYH"

[1] https://wiki.mikrotik.com/wiki/Manual:License

4.3.3 ARM nodes

The ARM computer chosen for our cluster is the Raspberry Pi 4 model B, represented *fig 4.3*. It runs over Raspberry Pi OS (formerly known as Raspbian) version Buster. It is based on the 5.10 version of the Linux Kernel.

This Raspberry uses an ARM-Cortex-A72 processor, which holds 4 computing cores.

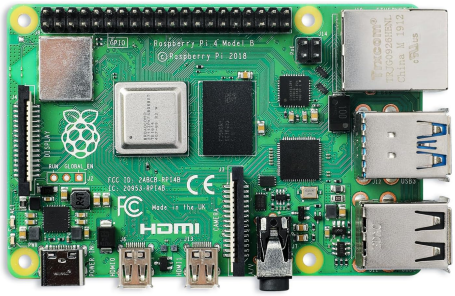

Figure 4.3: Raspberry Pi 4 model B

4.4 Cluster

An example of implementation of a cluster using our materials is illustrated on *figure 4.4*. As shown, the laptop managing the range and the different nodes are all connected to the router. The router is configured to give them fixed IP addresses for convenience, and an IP range has been designated for the range.

While everything is physically connected to the local network, only the different nodes are connected together in the Kubernetes network, which operates at a software level. The only access to this network outside of the node will be through the exposed services defined by our configuration. Our laptop is connected to the controller node with SSH to manage the cluster and launch or close the different scenarios.

As illustrated by this example, the hybrid design allows the worker node to be either X86-64 or ARM nodes, depending on our needs.

4.5 Creating a new scenario

Since the script to launch a scenario is a general purpose one, we need to respect a file structure in each new scenario for it to be compatible with our infrastructure. The file structure for scenario without using virtual machine, which is the minimal file structure for our cluster, is illustrated *fig 4.5*.

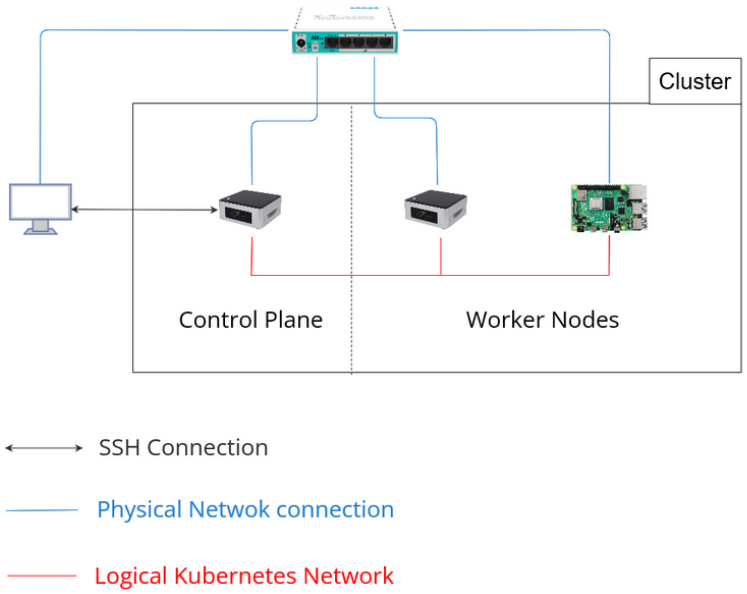

Figure 4.4: Example of implementation

Each scenario has at least two different folder that characterizes them: configs which holds the configuration files for each pods deployments and services which holds the configurations of the services exposing our applications. Two folders are optional: dockerfiles, which hold the dockerfile configuration of customized containers, and vm, which holds the necessary files to deploy VMIs.

Creating a new scenario thus begins with creating an appropriate folder in the scenario folder, containing at minima a "services" and a "configs" folders.

To add a component to the scenario, we need to create a folder with its name containing the necessary YAML configuration file in each of the folders.

We can easily re-use components from previous scenario by copying the folders corresponding to the component, with eventual minimal modifications (ip addresses, port, *etc*). Since we mostly use containers, it is possible to modify a container created in a previous scenario easily by creating an associated Dockerfile to build a new container on top of it.

Because of the inclusion of Kubervirt, we can also reuse existing virtual machine workload, by putting the different files for the VMI configuration in the "VM" folder.

4.6 Management

The management of the cluster and the cyber range by a trainer can be entirely done from his laptop, connected on the same router as the cluster. For this purpose, several scripts have been developed. Those scripts allow the set-up and the shutdown of the cluster, but also to launch scenarios on our range.

fig 4.5 illustrate the minimal file structure of our infrastructure. We created a Makefile in the root directory to handle the different command necessary for a trainer to deploy and manage both the cyber range and the scenario and make the calls to the different scripts from the script directory. If a trainer doesn't know all the commands, he can simply type the command "make" into his terminal and press the tabulation key for auto-completion and he will be shown the different commands.A README is available in the script directory for a more detailed explanation of the different commands.

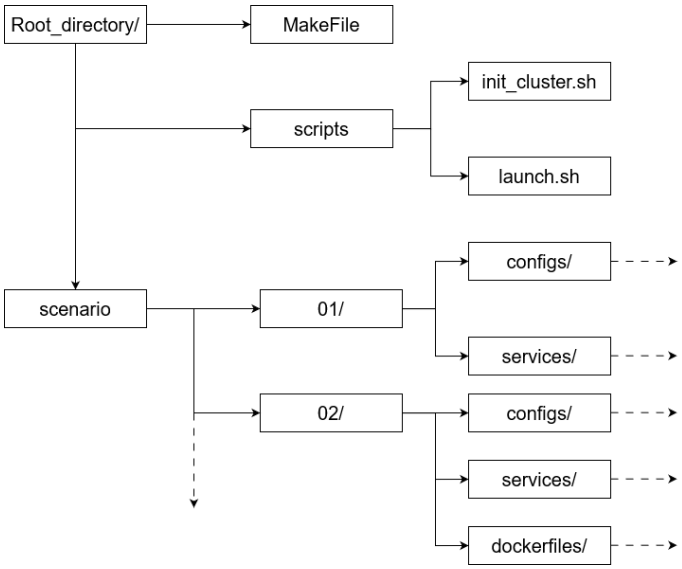

Figure 4.5: Minimal file structure for the infrastructure

4.7 Participating in a Scenario

Participants in a scenario receive a combination of <IP:PORT> per components of the scenario and are told what kind of component it corresponds to. While it would be very interesting to teach students about enumeration techniques. Our current implementation does not allow black box training, since we have to make

the network connection from outside pass by ports of our nodes, meaning that pods from a different namespace share a same IP (but with a different port).

Once a trainee gets the IP of the components, he simply needs to interact with them as if he would with actual physical machines. Depending on the scenario and the components, a trainee may have to interact with them using different protocols.

Chapter 5

Case Studies

This chapter aims to describe the scenario used as cases studies to test our infrastructure. The first scenario serve as a proof of concept that the infrastructure is able to runs at least simple applications, while the second serve as a proof of concept of the contributions of this master thesis.

5.1 Scenario 1: Damn Vulnerable Web Application

The scenario used Oh *et al.*in their 2020's paper [1] was taken as a first proof of concept of our infrastructure. The objective of this first scenario in the scope of this Master's thesis was to test if our infrastructure could replicate the use case showed by Oh *et al.*.

In this scenario we use the Damn Vulnerable Web Application (DVWA)[23], whose homepage is illustrated *fig 5.1*. DVWA is a vulnerable web application designed to teach common we vulnerabilities through different levels of difficulty. Some of its vulnerabilities were documented on the application itself to teach them directly while some of them were undocumented for the trainee to discover.

For the purpose of this scenario, we deploy pods with DVWA containers for each trainee.

5.2 Scenario 2: Famous Inc

The second scenario is meant to display the hybrid possibilities of our infrastructure. It holds both x86-64 and ARM workload and use different applications that the trainee need to access in order to complete the exercise.

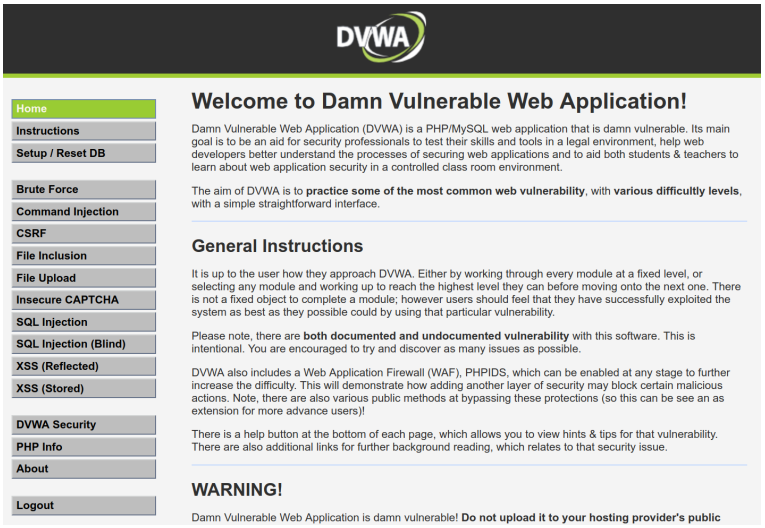

Figure 5.1: Damn Vulnerable Web Application's homepage as deployed on our infrastructure

We'll first give a high-level description of the scenario before explaining the different components involved in this scenario and describe how a trainee could solve the exercise.

5.2.1 High-Level Description

The second scenario put the trainee into the skin of a red team operator during an operation for a software company named "Famous Inc" who wanted to make sure their license server is secure.

Several assumptions are made in the challenge context. First, while the company has a firewall and the license server don't accept connection from outside of the local network, the trainee, representing the red teamer, has obtained the SSH access to the Raspberry Pi used by an employee connected to the enterprise network due to the usage of default credentials. Furthermore, the credentials to a file server containing the SSH key to the license server are stored on the raspberry pi. Finally, it is needed to connect to the raspberry pi to connect to both the file server and the license server since they are in the internal network of the company and reject foreign connections.

The objective of the trainee is to access the License server and obtain a license number.

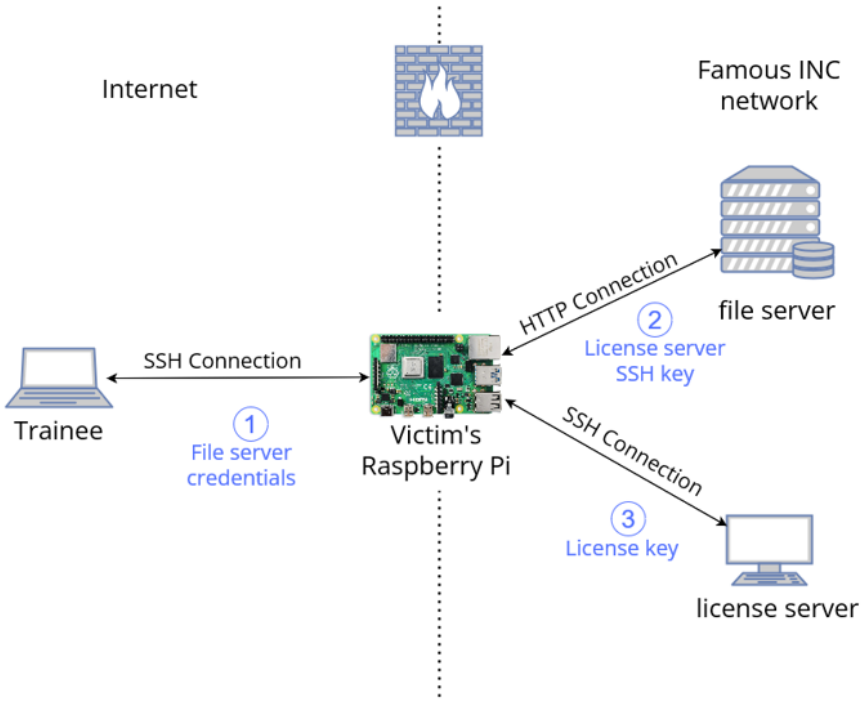

Figure 5.2: Scenario 2's infrastructure

5.2.2 Components

The second scenario, illustrated by *fig 5.2*, has 3 components inherent to the scenario, all simulated by our interface: the victim's Raspberry PI, the file server and the license server. We'll describe those components in this subsection

5.2.2.1 Raspberry Pi

The Raspberry Pi is represented in our infrastructure either by a Raspbian Stretch container if an ARM node is configured on the cluster, or an ARM VMI installed with Raspbian stretch instead. It is the only ARM workload on this scenario.

The Raspberry Pi component is configured to have 100 folders (10 folders containing 10 folders each) among which the credential information are randomly hidden. Because of that, each trainee will have a slightly different instance and a trainee participating in the exercise again will still have to do realize the step of searching for the file.

5.2.2.2 File Server

The file server is an Nginx container running over Linux Alpine. It is protected by a username and a password that can be found on the raspberry pi component.

It contains different files including the SSH key for the license server.

5.2.2.3 License server

The license server is a Linux Alpine container accessible over SSH. It contains a zip file protected by a password which contains the license key that the trainee has to find.

5.2.3 Solving the exercise

The basic steps to solve the exercise can be briefly summarized in three steps, represented in the *fig 5.2*. A more detailed step-by-step description can be found in the first appendix of this master thesis.

① The trainee need to connect to the raspberry instance through SSH. This will allow him to access the other component of the scenario, which will block traffic outside of the company network. In this step, he'll also need to obtain the credentials to the file server needed for step 2.

② Using the credentials obtained in the first step, the trainee need to make HTTP requests to the file server. This can be done either through a browser (in the case the user installs an RDP) or through CURL commands, for example. The objective in this step is to obtain the SSH key stored on the file server and give it the correct permissions.

③ Using the SSH key to connect to the license server, the trainee will find a single password protected zip file named license_keys. The user will need to crack the password of the zip file to finally complete his assignment.

Chapter 6

Validation

This chapter aims to validate our approach by testing different components of our design. We will first describe some proof of concepts we did to ensure basic functionalities, before testing the workload our implementation can support. After that, we will measure the time for the set-up of the implementation. Finally, we'll measure the different cost of our cyber range, by pricing its components and measuring the power consumption in different states and compare those cost to the costs of a cloud-based implementation of a similar infrastructure.

6.1 Proofs of Concept

We will first test that it is indeed possible to deploy a scenario on our cyber range. We will then check that our implementation allows us the modularity necessary to easily implement different scenarios, but also highlight its hybrid capabilities. Finally, we will prove isolation between instanced deployed for different trainees.

6.1.1 Deployment of scenarios

The fundamental role of a cyber range is to deploy scenarios. As such, the first proof of concept we needed was to prove that we could deploy scenario over our infrastructure. For that purpose, we decided to use the scenario used by Oh *et al.*[1] as a first case study, and use a container version of the DVWA, and developed a first deployment script.

A screenshot of the homepage of the DVWA as deployed on our infrastructure can be found *fig 5.1*, on page 24, and the deployment of the scenario itself, followed by the show of the deployed pod is illustrated *fig 6.1*.

```
jdesalle@thesis01:~/master-thesis$ echo "proof-of-concept">test.names
jdesalle@thesis01:~/master-thesis$ export SCENARIO=01
jdesalle@thesis01:~/master-thesis$ export NAMELIST=test.names
jdesalle@thesis01:~/master-thesis$ make general_launch_containers >/dev/null
jdesalle@thesis01:~/master-thesis$ kubectl get pods -n proof-of-concept
NAME                                 READY   STATUS    RESTARTS   AGE
proof-of-concept-5cb8564689-9j6jj    1/1     Running   0          11s
proof-of-concept-5cb8564689-hb5qx    1/1     Running   0          11s
proof-of-concept-5cb8564689-wrnhk    1/1     Running   0          11s
jdesalle@thesis01:~/master-thesis$
```

Figure 6.1: Deployment scenario 1

6.1.2 Modularity and hybridization

A significant characteristic of our infrastructure is its modularity, enabling the creation of new scenarios and the reuse of components across different training modules through the use of containers. Indeed containers are easy to modify and reuse in different contexts.

In order to demonstrate that ability, we developed a general file structure to follow and a general deployment script, allowing launching any scenario developed by following the specific file structure. We also created the second scenario, in 2 versions: one with only container (but needed an ARM node) and one using a VMI. *Fig 6.2* illustrate the deployment of a different scenario through the same script as in the first proof of concept.

Our tests showed two things about the modularity of our infrastructure: firstly

```
jdesalle@thesis01:~/master-thesis$ echo "modularity">test.names
jdesalle@thesis01:~/master-thesis$ export SCENARIO=02
jdesalle@thesis01:~/master-thesis$ make general_launch_containers >/dev/null
jdesalle@thesis01:~/master-thesis$ kubectl get pods -n modularity
NAME                                    READY   STATUS    RESTARTS   AGE
modularity-fileserver-766f447b59-4bhph  1/1     Running   0          16s
modularity-fileserver-766f447b59-85tdk  1/1     Running   0          16s
modularity-fileserver-766f447b59-jfszf  1/1     Running   0          16s
modularity-licence-6748446f88-9pr9b     1/1     Running   0          16s
modularity-licence-6748446f88-9zwz7     1/1     Running   0          16s
modularity-licence-6748446f88-sgd76     1/1     Running   0          17s
modularity-rasp-7bdb849b45-6s5x6        1/1     Running   0          15s
modularity-rasp-7bdb849b45-nw5kl        1/1     Running   0          15s
modularity-rasp-7bdb849b45-ptfnm        1/1     Running   0          15s
```

Figure 6.2: Deploying another scenario from the same script

if we follow the file structure the general-purpose script is indeed able to launch different scenario without modification, whether they include VMI or not. We can also reuse containers from a previous scenario to use in a new one, that is shown

by the 2 version of the second case study.

Another important characteristic highlighted by this proof of concept is the hybrid character of our cyber range implementation. Indeed, the 2 version of the second case study can illustrate both kinds of hybridization of our infrastructure: it uses both X86-64 and ARM components, but can also be deployed in a version that uses both containers and VMI components.

6.1.3 Isolation

Isolation between instances deployed for different trainee is crucial in the scope of a cyber range. Indeed, the action of a trainee shouldn't affect the training of other trainees, especially in a damaging way. In the Cas of our implementation, isolation is managed using Kubernetes namespaces. While that in itself should guarantee isolation, we decided to test it with a simple proof of concept. We launched the

```
jdesalle@thesis01:~/master-thesis$ kubectl get services --all-namespaces
NAMESPACE     NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)
        AGE
default       kubernetes            ClusterIP      10.96.0.1        <none>           443/TCP
        4m27s
isol1         ssh-service-file      LoadBalancer   10.105.211.36    192.168.88.252   2221:31875/TCP
        2m14s
isol1         ssh-service-licence   LoadBalancer   10.110.46.239    192.168.88.252   2220:31316/TCP
        2m15s
isol1         ssh-service-rasp      LoadBalancer   10.110.233.134   192.168.88.252   2222:31563/TCP
        2m13s
isol2         ssh-service-file      LoadBalancer   10.104.0.151     192.168.88.252   2221:32432/TCP
        2m8s
isol2         ssh-service-licence   LoadBalancer   10.110.230.67    192.168.88.252   2220:31271/TCP
        2m9s
isol2         ssh-service-rasp      LoadBalancer   10.104.177.59    192.168.88.252   2222:32686/TCP
        2m7s
kube-system   kube-dns              ClusterIP      10.96.0.10       <none>           53/UDP,53/TCP,91
53/TCP   4m24s
```

Figure 6.3: Service deployed for the proof of concepts

second scenario in 2 different namespaces. *fig 6.3* show the services deployed on our cluster during that proof of concept. As we can see, two different namespaces have been configured, test1 and test2, and while they share their external IP their port are different.

From our laptop, we connected separately to the raspberry component from the two name space. In one of the namespaces, we created a new directory named "test." When showing the directory available in each instance, it only appears in the instance it was created on and not in the other namespace. This process is illustrated *fig 6.4*.

Figure 6.4: Illustration of the isolation

6.2 Maximal workload Evaluation

In this section, we will test the maximum workload that our infrastructure can support. In the scope of the education, when we teach a class, we should be able to launch a scenario for several students for them to work on. We should also be able to scale our infrastructure in order to accommodate more students. In this section, we develop the results of a stress tests testing the maximum pod workload a container can deploy.

The script used for our stress test uses a particularity of Kubernetes to determines when we reach the maximal number of pods: once no resources are left to deploy a pod, that pod get stuck in a pending state indefinitely. We chose to assume the maximal number of pods is reached once a pod is stuck in a pending state for at least 5 minutes.

For the scope of this test, we chose to deploy pods containing the DVWA container as in the first case study as it is a complete application running in a single pod, making it a good benchmark, and it was already configured to be deployed on our infrastructure.

Maximal number of pods		
1 worker	2 worker	3 worker
68	134	195

Figure 6.5: result of the pods stress test

6.3 Infrastructure Setup Time Evaluation

This chapter aims to measures the time our infrastructure takes to be deployed and ready for a training session. We will first measure the time to create a cluster, before looking at the time of deployment of container-only scenarios and then the time of deployment of scenarios including VMIs. Finally, we will summarize the data gathered and evaluate the time it would take to set up a training exercise for a class of students. All the statistics related to the time measurment can be found in the second appendix.

6.3.1 Cluster creation

The first timing we will be measuring is the time it take to set up a cluster. This script is quite straightforward: we simply time the complete execution of the cluster initialization script, since the script finish once the cluster is initialized.

Fig6.6 illustrate the result of our script. We measured an average of 45.852 s to initialize the cluster. The median time is 46.365 s and the standard error is 0.715 s.

Figure 6.6:

6.3.2 Deployment of containers-only scenarios

A second measurement is the time to launch scenario containing only containers: we will measure both the first and the second case study, to compare the scenario with more components. The script we made take two arguments: the number of iterations and the name of the scenario. For the purposes of this stress test, each scenario was timed 20 times. The script uses the bash "time" command to measure the time it takes to launch the scenario and to have no state other than the accepted state when checking the pod deployment statute.

Concerning the deployment of the first scenario, *fig6.7* illustrate the result of our script. We measured an average of 6.6075 s to initialize the cluster. The median time is 6.095 sand the standard error is 0.256 s.

Concerning the deployment of the first scenario, *fig6.8* illustrate the result of our script. We measured an average of 12.1705 s to initialize the cluster. The median time is 12.27 s and the standard error is 0.256s.

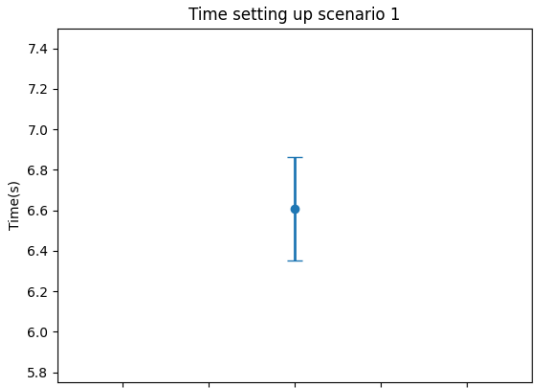

Figure 6.7:

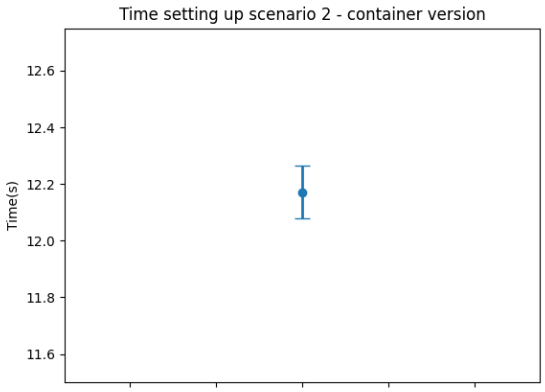

Figure 6.8:

6.3.3 Deployment of VMI-incorporated scenario

The Last measurement is the time to launch scenario containing a VMI: we will used the second case study in his VMI version to ev+aluate that time. The script we made take two arguments: the number of iterations and the name of the scenario.

Fig6.6 illustrate the result of our script. We measured an average of 81.8955 s to initialize the cluster. The median time is 81.25 s and the standard error is .6558 s.

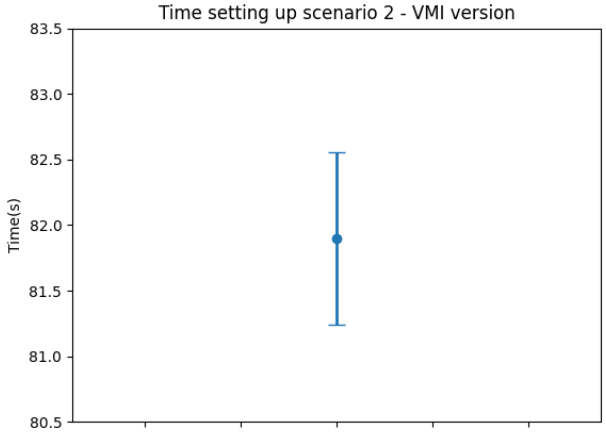

Figure 6.9:

6.3.4 Setting up a training class

Considering a time of about 3-4 minute to install the material and to power the Nodes, we have thus around 45 second to initialize the cluster and maximum 80 second to deploy a scenario. The total time is lower than 10 minute, which seems like a reasonable time for a set up at the beginning of a class session.

6.4 Costs Evaluation

In this section, we will assess the cost of our infrastructure by estimating the cost of buying and maintaining our infrastructure by looking at the cost of the materials and measuring the power consumption.

6.4.1 Material costs

For the purposes of this study, we will consider the cost of an infrastructure identical to the one used on our tests. One of the advantage of our framework is that it can be used with older material that is left from other project, which mean the cost of the materials can sometimes be entirely offset.

For our test we used: 4 intel nucs, 1 raspberry pi 4 model B, and our Microtik Hex Lite router.

The model of our NUC is unfortunately discontinued, and cant be bought new. For second hand model, it can be easily found between 130 and 140€ per unit.

The raspberry Pi that we used is currently sold 69,95 € without cable or case. We will consider those aditional cost negligible.

The Microtik Hex Lite router is currently sold around 50€.

For the total infrastructure, we thus have a material cost of approximately 680 euro for our total infrastructure.

6.4.2 Power Consumption

In order to measure the consumption of our infrastructure, we used a wattmeter from the company "fishtech", illustrated *fig 6.10*.

The result of our measurements are summarized *fig 6.11*. We measured our power consumption at 3 different states: when everything is connected but the cluster not initialized (refered to as "idle", when the cluster is initialised, and when the workload is at its maximum (according to our maximal number of pods evaluation). For each of these situation, we evaluated the situation with one, two or three worker nodes connected, in addition to our control node and our router.

A maximum of 35.3 watt is used by our infrastructure, while the average power usage is closer to 20 watt, which is not too much for such an infrastructure

Figure 6.10: wattmeter used for our measurments

Idle			Cluster initialised		
1 worker	2 worker	3 worker	1 worker	2 worker	3 worker
12.8 W	15 W	17.9W	13.7 W	19.2 W	24.6 W
Maximal pod number					
1 worker	2 worker	3 worker			
20.7 W	28.1 W	35.3 W			

Figure 6.11: result of our power consumption tests

Chapter 7

Future work

This chapter explores possible paths to expands on the infrastructure presented in this master thesis. While it would be impossible to be exhaustive about all the possibilities to extend such a work, the solutions we present in this section aim to improve on some inherent limits of our infrastructure. We will first discuss the implementation of an actual load balancer service with MetalLB before discussing stronger means of isolation with OpenVPN.

7.1 MetalLB

A weakness of our current infrastructure is the handling of the networking from outside of the cluster. Currently, we need to pass through the IP address of a node of a cluster and encode it in the service configuration. It can't allow a complete black box training, and don't allow a trainer to teach correctly about enumeration techniques.

A solution would be to implement a complete load balancer service through a Kubernetes add-on made for bare metal such as metalLB. MetalLB [24] is an implementation of load balancer services for bare metal cluster.

Such an addition to our infrastructure would allow the trainer to reserve a range of IP addresses on the router, and distribute those addresses to each services directly through the configuration of the metalLB load balancer. Each service would thus have its own IP and only the port related to that service would appear open on that IP address. Each trainee could thus receive such an IP range where only his instance of the scenario would be deployed allowing the training of enumeration techniques over that IP range. It would also make penetration testing like scenario more realistic since IP ranges are commonly part of the scope defined in a pentesting engagement.

7.2 OpenVPN

The current isolation of our infrastructure guarantee that the action of a trainee on his instance will not affect the training of another trainee. But it doesn't stop a trainee from deliberately trying to interfere with another trainee's training. A solution to that problem would be an access control mechanism forbidding a trainee to interact with the <IP:Port> combination from another student.

A solution may be to use virtual private networks(VPN), for example using a tool such as openVPN. Each student would thus have a private network including all the deployed components from the scenario they would be playing. Such a solution would prevent student from interfering with other students' training since they would not be able to access or see any deployed scenario not intended for their own usage.

7.3 Scenario randomization

If a student need to repeat a scenario, it may be interesting to be able to add some randomness to make him learn different additional skills compared to his first pass. While we had some small mechanism in place to vary the experience, like in the raspberry pi component of the second case study, where the login information for the file server are randomnly pu in a different directory each time the scenario is deployed, we don't have a mechanism that would allow the student to try something different.

For container based workload, an approach like the one propose by Nakata *et al.*in their CyExec white paper [3] may be an interesting direction to improve on that aspect.

Chapter 8

Conclusion

In this Master's thesis, we described an hybrid cyber range design and an analysis of its cost and performances. Our analysis and implementation answered both our research question.

In the scope of this Master's thesis, several contribution were realized:
First, we designed a framework to deploy an hybrid cyber range over a cluster of compute. We developed an approach allowing us to use both X86-64 and ARM workload in our cyber range, and to deploy bot containers and virtual machines.

Second, we generalized our approach to have a modular design, allowing to to easily modify an existing scenario, reuse part of a scenario or even create a new scenario.

Third, we confirmed this approach is scalable: adding new nodes to our cluster allow us to deploy more workload and to cather to more students in a same class.

A a last contribution, we confirmed the approach is cost effective and that the cost of our infrastructure doesn't seem to much for an education perspective. Th time our installation take to set up from the beginning to having a running scenario also align with a usage in an education setting.

finally, We also identified some limitation in our current design and identified some direction a futur work could follow to improve on this Master's thesis, but such a work is always a work in progress as a perfect infrastructure is not achievable.

Bibliography

[1] S. K. Oh, N. Stickney, D. Hawthorne, and S. J. Matthews, "Teaching web-attacks on a raspberry pi cyber range/," in *Proceedings of the 21st Annual Conference on Information Technology Education*, pp. 324–329, 2020.

[2] N. Jain and S. Choudhary, "Overview of virtualization in cloud computing," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pp. 1–4, 2016.

[3] R. Nakata and A. Otsuka, "Cyexec*: A high-performance container-based cyber range with scenario randomization," *IEEE Access*, vol. 9, pp. 109095–109114, 2021.

[4] F. Rodríguez-Haro, F. Freitag, L. Navarro, E. Hernánchez-sánchez, N. Farías-Mendoza, J. A. Guerrero-Ibáñez, and A. González-Potes, "A summary of virtualization techniques," *Procedia Technology*, vol. 3, pp. 267–272, 2012.

[5] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support paas," in *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614, IEEE, 2014.

[6] "chroot(2) - linux man page." https://man7.org/linux/man-pages/man2/chroot.2.html *Accessed on April, 2023.*

[7] "namespaces(7) — linux manual page." https://man7.org/linux/man-pages/man7/namespaces.7.html *Accessed on April, 2023.*

[8] "cgroups(7) — linux manual page." https://man7.org/linux/man-pages/man7/cgroups.7.html *Accessed on April, 2023.*

[9] "Docker documentation." https://https://docs.docker.com *Accessed on November, 2022.*

[10] R. Nakata and A. Otsuka, "Evaluation of vulnerability reproducibility in container-based cyber range," *arXiv preprint arXiv:2010.16024*, 2020.

[11] "Kubernetes documentation." https://kubernetes.io/docs *Accessed on March, 2022.*

[12] "Kubevirts documentation." https://kubevirt.io/user-guide/ *Accessed on March, 2022.*

[13] "Libvirts documentation." https://libvirt.org/docs.html *Accessed on April, 2023.*

[14] "Qemu documentation." https://www.qemu.org/documentation/ *Accessed on April, 2023.*

[15] N. Chouliaras, G. Kittes, I. Kantzavelou, L. Maglaras, G. Pantziou, and M. A. Ferrag, "Cyber ranges and testbeds for education, training, and research," *Applied Sciences*, vol. 11, no. 4, p. 1809, 2021.

[16] R. Beuran, C. Pham, D. Tang, K.-i. Chinen, Y. Tan, and Y. Shinoda, "Cytrone: An integrated cybersecurity training framework," 2017.

[17] C. Pham, D. Tang, K.-i. Chinen, and R. Beuran, "Cyris: A cyber range instantiation system for facilitating security training," in *Proceedings of the 7th Symposium on Information and Communication Technology*, pp. 251–258, 2016.

[18] T. Debatty and W. Mees, "Building a cyber range for training cyberdefense situation awareness," in *2019 International Conference on Military Communications and Information Systems (ICMCIS)*, pp. 1–6, IEEE, 2019.

[19] P. Legg, A. Mills, and I. Johnson, "Teaching offensive and defensive cyber security in schools using a raspberry pi cyber range," in *Journal of The Colloquium for Information Systems Security Education*, vol. 10, pp. 9–9, 2023.

[20] R. Beuran, T. Inoue, Y. Tan, and Y. Shinoda, "Realistic cybersecurity training via scenario progression management," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 67–76, IEEE, 2019.

[21] W. Crumpler and J. A. Lewis, *The cybersecurity workforce gap.* JSTOR, 2019.

[22] "Routeros documentation." https://help.mikrotik.com/docs/display/ROS/RouterOS/ *Accessed on July, 2023.*

[23] "Damn vulnerable web application." https://github.com/digininja/DVWA *Accessed on April, 2023.*

[24] "metallb documentation." https://metallb.universe.tf/ *Accessed on June, 2023.*

Appendix

Appendix I: Solving Case Study 2 - step-by-step procedure

The trainee receive 3 <IP:PORT> combination, corresponding to the different machines on the network The one corresponding to the raspberry is identified for him, and he's told that the machine isn't secure because of a default password, and because ssh is open to the Internet.

The first step is to connect to the raspberry pi container

```
$ ssh pi@<raspIP> -p <raspPORT>
```

The password is the default raspbian password "raspberry"

The trainee is, now in the raspberry and need to find information about the licence server that may have been stored on the raspberry pi.There's a lot of document, but we can use grep to fin about that information.

```
$ grep -r licence .
$ cat <fileFoundByGrep>
```

In this file, we get the password and the username for the file server. Since we don't use a full load balancer service , we can't really use Nmap to identify them. For this reason, each machine is identified for the trainee, and the trainee should know which <IP:PORT> correspond to the file server.

Once conneted to the fileserver, the trainee can download the "id_rsa". However it is not sufficient to be used directly to connect to the licence server: we need to give the correct access right to the key:

```
$ chmod 600 id_rsa
```

It is now possible connect to the validator server using the SSH key:

```
$ ssh famousAdmin@<licenceIP> -p <licencePORT> -i id_rsa
```

Once in the validator server the trainee will see a licence.zip that he will need to exfiltrate The two easiest way should be through SCP or netcat. We'll use netcat in the scope of this write up.
From the computer of the trainee, we need to set up a netcat listener on port 1337 that will store the file:

```
$ nc -lnvp 1337 > licence.zip
```

From the validator server, we'll need to send the file through netcat to the listenner on port 1337:

```
$ nc -w 3 <userIP> 1337 < licence.zip
```

When the trainee will try to unzip the file, he will discover the file is protected by a password: we need to bruteforce that password

First step is to recover the hash of the password, we'll use zip2john

```
$ zip2john licence.zip >hash_licence
```

John the ripper can finally be used to bruteforce the password using our wordlist

```
$ john --format=pkzip hash_licence -wordlist=<pathToWordlist>
```

We thus obtain the licence key.

Appendix II: time statistics from validation

Time cluster initialisation

values:[48.21, 46.38, 43.48, 44.11, 39.43, 47.3, 48.21, 50.44, 45.77, 45.28, 49.19, 47.91, 42.5, 46.98, 49.45, 41.47, 49.85, 46.35, 40.27, 44.46]
mean:45.852
median:46.365
min:39.43
max:50.44
first quartile:43.9525
third quartile:48.21
interquartile spread:4.2575
std error: 0.7147769619001144

Time scenario 1

values:[6.07, 7.61, 6.3, 7.83, 5.97, 6.1, 5.9, 5.94, 6.15, 6.08, 7.59, 6.09, 5.89, 7.53, 6.0, 6.3, 6.02, 6.0, 6.17, 10.61]

mean:6.6075
median:6.095
min:5.89
max:10.61
first quartile:6.0
third quartile:6.6075
interquartile spread:0.6074999999999999
std error: 0.2563343571033328

Time scenario 2 - VMI version

values:[81.17, 81.33, 87.32, 82.68, 84.51, 78.36, 87.0, 78.96, 82.12, 83.08, 84.14, 78.8, 79.48, 83.04, 80.09, 87.29, 78.45, 79.76, 80.67, 79.66]
mean:81.8955
median:81.25
min:78.36
max:87.32
first quartile:79.615
third quartile:83.345
interquartile spread:3.730000000000004
std error: 0.6557532731623671

Time scenario 2 - Container version

values:[12.47, 12.13, 12.32, 12.28, 12.51, 10.76, 12.6, 12.46, 11.42, 12.18, 12.04, 12.3, 12.18, 12.44, 12.39, 12.29, 12.25, 12.16, 11.97, 12.26]
mean:12.1705
median:12.27
min:10.76
max:12.6
first quartile:12.1525
third quartile:12.4025
interquartile spread:0.25
std error: 0.09298691192567742

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**