Automation of Cyber Range Federation Using KYPO CRP

Snellings Matteo

Master thesis submitted under the supervision of
Prof. Wim Mees

in order to be awarded the Degree of
Master in Cybersecurity
Cryptanalysis and Forensics

Academic year
2022 – 2023

I hereby confirm that this thesis was written independently by myself without the use of any sources beyond those cited, and all passages and ideas taken from other sources are cited accordingly.

30/05/2023

Title: Automation of Cyber Range Federation Using KYPO CRP

Author: Snellings Matteo
Master in Cybersecurity – Cryptanalysis and Forensics
Academic year: 2022 – 2023

Abstract

We are living in a world where the internet is playing a big role in our lives. Due to this increase usage inside companies, but also for individuals, there has also been an increase in threats that appear over the internet. It is important to be able to counter these threats. A very important step in doing so, is to be able to understand what the vulnerabilities are, and how to protect from them. In the last decades, a tool that is more and more used in order to achieve this goal is a cyber range. This allows the simulation of attacks in a completely sealed environment. Using such technology is needed in order to train the different security specialists, that will be in charge of defending the network for incoming threats. One of such technologies is KYPO CRP, an open-source platform developed by the Masaryk University.

Using a single instance of a cyber range is often not enough, as we would like to have access to more specialized documents and technologies, such as, for example, medical records that are stored on a specific location, a simulation of a satellite that will be used by company and is also located remotely,... All these examples, make it a necessity to inter-connect multiple cyber ranges together. This is called federation of cyber ranges. The goal of this work will be to develop a way, in KYPO CRP, to automate the process of inter-connecting the ranges together.

Keywords: Cyber Range, KYPO, OpenVPN, Federation

Acknowledgements

This master thesis marks the end of my Master's degree in Cybersecurity. During that time, I had the opportunity to acquire new knowledge, meet new people making this an incredible journey. The completion of this paper could not have been possible without the help of the following people to whom I would like to express my gratitude.

First of all, I would like to thank my academic supervisor, Prof. Wim Mees who was always able to find time to answer my questions. His advise throughout the development of this work has been very precious.

I would also like to thank Prof. Georgi Nikolov and Prof. Thibault Debatty who accepted to read this thesis and will be part of my jury during the defense of this work.

The next people I would like to thank are my friends for their support throughout these two years. Especially Adrien Defer, Arnaud Demeure and Loïc Caudron, with whom I have exchanged a lot on this work and that allowed me to keep motivated in order to finish this master thesis.

Finally, I would like to thank my family. Without their support the completion of this thesis would not have been possible. I am very grateful for the interest they showed in my work as well as their love and care that made it possible for me to be where I am right now, even though it is already my second master's degree, making it a seven year long journey.

Table of Contents

List of Figures

Chapter 1
Introduction

The internet is becoming more and more important in today's society as over 60% of the world population currently has access to it [18]. This is even more true when looking at internet usage inside the enterprise world.

Due to the high number of data that is processed every day and the speed that is needed to perform different types of computations, enterprises often work with a cloud service provider. This means that their data will be stored in a remote location, managed by the cloud service provider, and accessed over the internet. We can see that over 90% of the businesses are using some sort of cloud service provider, confirming that it has become a great part of the business [2].

A second aspect of internet usage in enterprises is the use of VPN or "Virtual Private Networks". This technology allows employees to access the internal network from anywhere in the world. VPNs have become even more used in the last few years due to the recent pandemic that has occurred and forced people to adapt and start working remotely. In only 3 years the market share for VPN usage in businesses has almost doubled in value [17].

Lastly, the employees that are inside of the company's network also need access to the internet for various reasons: personal use, if it is allowed by the company policies, research on work-related topics,... The company could also host its web server or mail server which would need to be accessible from the outside of the network.

Taking this all together confirms that the internet is playing a huge role in our personal lives, but also in the business world. However, an increased usage of the internet also comes with a bigger issue: attackers can try to leverage more vulnerabilities to get inside the network. With more interactions being made over the internet (remote data storage, remote access/VPN, internet browsing,...), the number of potential vulnerabilities, and therefore also the attack surface, increases. This means that a lot of effort should be taken by the different enterprises to reduce this attack surface as much as possible. It is not possible to ensure that a system is secure at 100%, therefore the relevant teams responsible for the company's security must be able to react swiftly when and if such incidents occur. To achieve this, we need to be resilient in our approach to treating cybersecurity threats. This means that different security measures need to be implemented knowing that some attacks will occur.

The definition of cyber resiliency from the NIST institute is the following [16]:

"*The ability to anticipate, withstand, recover from, and adapt to adverse conditions, stresses, attacks, or compromises on systems that use or are enabled by cyber resources. Cyber resiliency is intended to enable mission or business objectives that depend on cyber resources to be achieved in a contested cyber environment.*"

One critical aspect that is needed to build good resiliency, is the concept of situation awareness. This awareness is needed to be able to take quick and efficient actions when there is an attack or data breach that has been detected. There exists a model that has been discussed in [4] that uses three different levels of situation awareness that need to be present to be able to perform good decision-making. Those steps can be summarized with the diagram shown in Figure 1.1

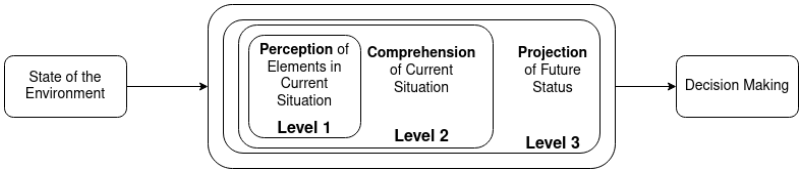


Figure 1.1: Situation Awareness as described in [4]

However, as explained in [3], it is not enough to train the situation awareness of only one person in the company. This is because there are many reasons why the different levels of situation awareness of a single person might fail [9]. To prevent this single point of failure, we will need to train an entire team on their situation awareness and decision-making. This is where the Cyber Ranges have been introduced.

1.1 Cyber Ranges

A cyber range provides a complete virtualized environment that can be used for training. The goal is to have a virtualized network that is as close as possible to the real one that needs to be secured. By doing so, a team of security professionals will be able to access the cyber range and simulate a scenario where there is an ongoing attack.

Cyber ranges are a novel technology and can be classified into 4 main groups [11] [8]:

- **Simulation Ranges** – This kind of cyber range will virtualize every component based on their behavior in the real network. This means that every machine, every server, and every network component will be virtualized without the need of having a physical device. Those kinds of cyber ranges require a very strong orchestration layer. This layer will manage all interactions between the different machines. Since we will virtualize every aspect of the network, we will need lots of machines that need to be coordinated perfectly.

  The advantages and disadvantages of using this type of technology are:

  + Quick to spin-up;
  + High re-configuration speed;
  + High level of customization to match the real environment;
  − Unpredictable and unrealistic latency and jitter of network performance that is added due to the virtualization;
  − Lots of configuration.

- **Overlay Ranges** – This kind of cyber range runs as a virtualization layer on top of the existing hardware. Due to the use of the physical hardware, by using overlay

ranges, it is possible to have a high-fidelity replication of the actual network. This however comes at a cost mainly due to the hardware that needs to be used.

The advantages and disadvantages of using this type of technology are:

+ Very close to the real environment with high fidelity;
− High cost due to hardware;
− Potential compromise on network components if using a big network with limited resources.

- **Emulation Ranges** – For this type of ranges, dedicated network infrastructure is used. From there, there will be a mapping of the network, servers, and storage onto a physical infrastructure. This yields a closed-off environment where the user can test many different scenarios. It is also easy to add additional components, such as a DNS resolver, traffic generator,... In those environments it will also be possible to add user emulation, meaning that we will emulate traffic coming from the different workstations that model as accurately as possible the behavior of a real user.

  The difference between simulation and emulation could be challenging to comprehend. This difference could be described as [5]:

  "*[...]**emulation** is the process of enabling a computer system (the host) to mimic the hardware and software features of another target device (the guest). In other words, emulation aims to imitate the behavior of a computer or other electronic system with the help of another type of system. A **simulation** on the other hand is the process of modeling an environment to mimic the behavior and configuration of another target device. Compared to emulation, simulation is a lower-fidelity replica of the real system usually used for the purposes of analysis and study rather than for high-fidelity interaction.*"

  The advantages and disadvantages of using this type of technology are:

  + Very realistic testing scenarios;
  + Realistic internet simulation possible;
  − Need of specialized equipment.

- **Hybrid Ranges** – These ranges are customized combinations of the three types shown above.

  The advantages and disadvantages of using this type of technology are:

  + Highly customizable;
  + You can create the exact environment you want where some elements could be less accurate than others and therefore choose different types of cyber ranges for each of them;
  − Challenging to maintain;
  − Challenging to modify to add or remove a component.

1.2 Purpose of this work

This work has been realized in collaboration with Cylab, the cybersecurity lab of the Royal Military Academy[1]. The goal will be to analyze a new cyber-range technology called KYPO CRP. This is an open-source cyber range platform developed by the Masaryk University in the Czech Republic since 2013. It has been released only four years later in 2017. It uses a state-of-the-art approach to build its technology such as containers, micro-services, and many more.

One interesting feature is that KYPO also provides a lite version of its cyber range platform. This means that it could be possible to first develop an easy scenario on the lite version, study the feasibility and only when we want to scale it up, put it in the real environment. The focus of this work will be on the implementation of federation inside of KYPO Lite. The goal will be to be able to inter-connect multiple cyber ranges in a way that is as automated as possible

1.3 Organization of this document

We will first talk about the state-of-the-art regarding cyber ranges and the technologies that will be used in this work. Then, we will lay down the objectives and show how to implement them in the KYPO cyber range platform. We will continue with the results on an actual instance of KYPO lite. Finally, we will end this work with a conclusion on the topics that will be discussed.

[1] `https://cylab.be/`

Chapter 2
Literature review, state of the art (SotA)

In this chapter, we will discuss the state-of-the-art concerning cyber ranges. There are many different implementations of cyber ranges. We will go over their main characteristics as well as their general architecture. We will then conclude this chapter by presenting two newer cyber range platforms that exist.

2.1 Cyber ranges

As discussed in the previous chapter, it is essential to train the situation awareness of the team responsible for security. One very useful tool to do so is the cyber range. Cyber ranges are platforms where many users can connect at once and take control of a part of a simulated network. The network can be specified by creating a scenario. When doing so, we could deliberately compromise some hosts or rather make an exact copy of the existing network. The first approach helps learn new concepts and train cybersecurity skills interactively. The second approach is rather to be able to find new vulnerabilities that could be exploited as well as implement new protective measures inside the network that is implemented in reality.

When using a scenario, there are at least four teams that are necessary [22]:

- **Red Team** – A group of people that will emulate an actual adversary and try to attack the network;
- **Blue Team** – They are responsible for defending against the attackers;
- **White Team** – They design the exercise, make up the rules and evaluation criteria;
- **Green Team** – Responsible for the monitoring of the infrastructure during the exercise.

These teams are the ones that are used in most cases. We can however include two other teams as well:

- **Purple Team** – A mix of Red and Blue teams. They will simulate attacks on a system as well as propose solutions that would be needed to implement in order to patch the vulnerabilities they have found;
- **Yellow Team** – They are responsible for fixing the vulnerabilities that the Red team has found inside specific software.

Cyber ranges are built on top of virtualization. The amount of virtualization needed will depend on the use case and therefore also on the type of cyber range that is used. The four main types that have been listed in Chapter 1 are:

- **Simulation Ranges** – Simulation of all the components inside the network;
- **Overlay Ranges** – Simulation layer on top of physical hardware
- **Emulation Ranges** – Emulation of the network components by physical hardware;
- **Hybrid Ranges** – Combination of the three types above.

There are however other characteristics that can be used to classify different cyber range solutions [19] [15]:

- **Application Domain** – For which domain is the cyber range deployed: Academic, Military & Defense, Open Source,...?
- **Infrastructure Association** – What type of infrastructure is used: public, private, or federated?
- **Deployment** – What type of platform does the cyber range use: cloud platform, VPN, or no cloud infrastructure at all?
- **Team Formation** What teams does the platform support: red team, blue team, white team, green team, purple team?
- **Deployed Technology** What kind of technology does the cyber range use: virtual machines, sandboxes, containers,...?

More specifically in a Capture The Flag type of situation, where there is a rewarding system in place, [23] adds to those characteristics:

- **Scenario** – Defines if there is a story-line, the application domain of the scenario (IoT, cloud,...), the type (static or dynamic),...
- **Monitoring** – The methods, tools, and layers at which the monitoring is performed
- **Scoring** – The method and tools that are used to implement a scoring mechanism. The method refers to how the points are given (when achieving an objective, by looking at the logs,...). The tools that are used could be flag submission dashboards, log analyzers,...
- **Management** – Defines methods tools and techniques for the assignment of roles to the different individuals, the management of the resources, and the management of the whole cyber range.

2.2 Architecture

Depending on the implementation, the choice in the architecture of a cyber range might vary slightly. However, the overall architecture of systems that implement a cyber range will stay the same. First, we will present the overall architecture, which can be seen in Figure 2.1.

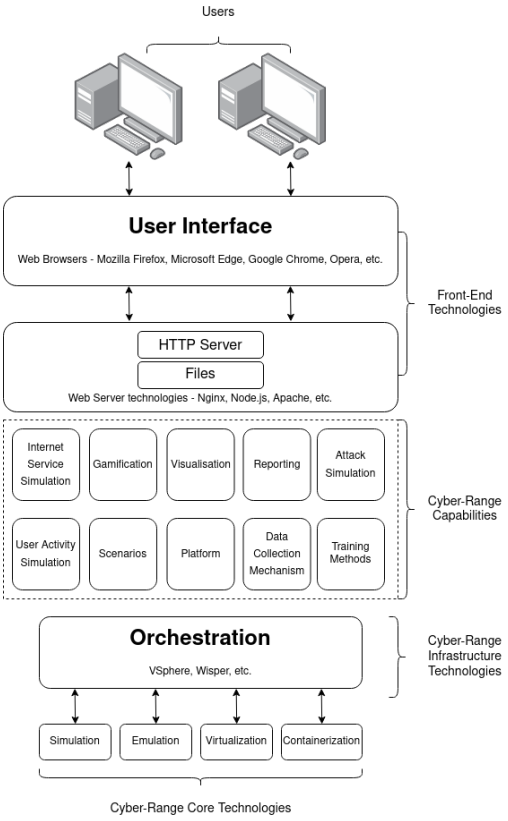


Figure 2.1: Cyber Range Architecture [19]

This higher-lever view shows that most of the cyber ranges are accessed over the internet using various front-end technologies. The actual software will be stored on servers that are put in place for this purpose.

The architecture of the cyber range systems relies on a virtualization layer. This is the bottom layer of the architecture, as depicted in the figure above. The elements that will vary in the diagram are the core technologies that are used. This will be dependent on the type (simulation, emulation, hybrid, or overlay) of the cyber range that is implemented. It is also possible to have a combination of those different technologies as well as a direct interaction with physical hardware.

The core technologies that will be used will then be coordinated by an orchestrator. This technology is most often developed differently for each cyber range [11]. This is because it will highly depend on the type of virtualization as well as the underlying infrastructure that is used. This underlying infrastructure is the set of servers, storage, and network on top of which the system will be designed.

A more precise view of the cyber range part of the architecture as described by the NIST institute in [11] can be seen in Figure 2.2.

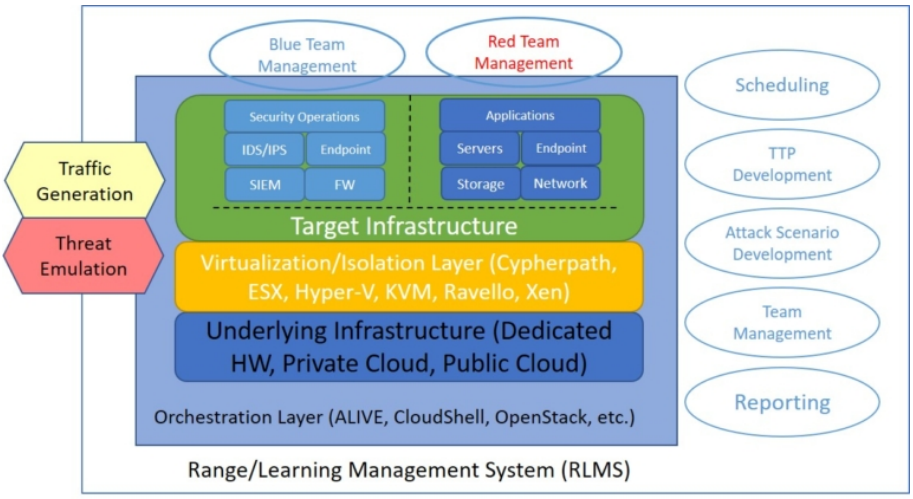


Figure 2.2: Cyber Range Architecture as described by [11]

Using this view, a Range Learning Management System (RLMS) is defined, which contains the features of a Learning Management System, more specifically for cyber ranges. According to [20], a Learning Management System can be defined as:

"*A learning management system (LMS) is a software that is designed specifically to create, distribute, and manage the delivery of educational content.*"

The Orchestration layer can be found back at the outer layer, containing the target infrastructure that has been built on top of the underlying infrastructure with the intermediary of a virtualization or isolation layer. The choice of the virtualization layer is very important as this could affect whether the cyber range represents the target infrastructure correctly or not. Furthermore, since the goal is to create a vulnerable network, the virtualization layer will make sure that there is no contact possible with the outside world (depending on the type of cyber range that is wanted). It will therefore act as a kind of firewall between the target infrastructure and the underlying infrastructure.

2.3 Existing Cyber Ranges

There are a lot of cyber ranges which are already deployed worldwide. Some of them are private, others are public. Detailed comparisons between different ranges have already been published: [19] compares a total of 44 cyber ranges using a detailed systematic approach, [23] performs a literature review of 100 cyber ranges and security test-beds, and [15] reviews 20 ranges as part of a master thesis work. We will therefore not make a detailed comparison of the existing ranges, as it has already been done. However, we will take the important points out of those articles as well as look at more recent solutions, such as KYPO CRP, which will be used further during this work.

2.3.1 Summary of articles

From [19], we can view that the overall number of publications on the topic of cyber ranges increases every year. This is what is to be expected due to the increasing number of cyber-attacks and therefore also an increasing need for the security of the different systems.

When classifying the different ranges that have been analyzed following different criteria we obtain:

- **Application Domain** – The main domain of activity where cyber ranges are used is in education and research for academic purposes (at 31%), followed by an equal amount in commercial organizations and Military Defence & Intelligence (at 24% each). The last 21% are divided between the service providers, government, open source, and law enforcement domains;
- **Type** – Most of the ranges that have been analyzed are for public and federated use (at 30% each), followed closely by private use (at 24%). The rest use combinations between public, private, and federated;
- **Teams Formations** – 67 % of the ranges use only a red and blue team configuration, where only read and blue team objectives are defined. 9% include the grey team as well, and the other 8% slices include either the green, purple, or white and yellow teams.
- **Experimentation Methods** – Half of the cyber ranges are of the simulation type, while 38% prefer to use emulation. The remaining 12% are divided between overlay ranges and hybrid ranges.

A second review of the literature can be found in [23]. The authors define 5 key aspects on which they will classify the cyber ranges. These have been discussed previously:

- **Scenarios** – Classified by their purpose, environment, storyline, type, domain, and tools;
- **Monitoring** – Classified by the different methods, tools and layers that are used;
- **Teaming** – Being a combination of red, blue, white and green teams;
- **Scoring** – Classified by the methods and tools used for that purpose;
- **Management** – Classified by the role, resource and range management.

When it comes to the classes defined above, 94% include details about the scenarios that can be used. 91% of papers also include details about the management of the different aspects of the cyber range. 86% contain elements about the monitoring infrastructure. Less than half of the papers talk about the different teams and only a quarter detail the scoring mechanism.

Let us now analyze the different classes with their specificities as described by the authors in [23]:

- **Scenarios** – Most papers describe a way of creating and deploying new scenarios. Some of them include event-generation features to make them more dynamic and realistic. The way of creating the scenarios is either done by using a dashboard or by using machine-readable languages such as XML or JSON. The code will then be read and compiled into different scenarios;
- **Monitoring** – Multiple ways are used to monitor the system as a whole. Some ranges use event logging and analysis while others employ other data collection mechanisms where the data will then be analyzed using different methods. The data is then displayed in a dashboard. The layers that are analyzed are mainly the application layers. Other papers prefer to inspect the network layer instead when it comes to monitoring;
- **Teaming** – The main types of teams that are used are the red, blue, white and green teams;
- **Management** – The management of the cyber range happens either by using a dashboard or by using specialized portals that allow communication directly with

the range. Most ranges that include this aspect have data storage modules that allow storing the different configurations, scenarios, tools that are used, rules,...

[15] analyzes different cyber ranges. We can first observe that most of the cyber ranges are deployed using a cloud infrastructure. Inside this cloud infrastructure, it is very common for those ranges to deploy Virtual Private Network (VPN). All of them support blue and red teaming. Only a few of them, however, support other teams such as the green, yellow or purple teams.

Another way that is used to classify the ranges is to see the type of isolation that is used when the range is deployed. Most of the ranges only run inside a virtual machine. Some of them, however, also have the ability to use sandboxes, which are more lightweight and can provide different levels of isolation.

2.3.2 New technologies

This section of the analysis of the state of the art will focus on more recent articles that present cyber ranges. We will put focus on two particular ranges: CyExec* and KYPO CRP as these technologies use novel techniques to achieve their goal.

For CyExec*, containerization is used instead of the more classical virtual machine-based approach. In the hypervisor-type environment, the hypervisor will create the guest OS. Every time a new virtual machine needs to be created, the entire environment is replicated, which could lead to a high load and therefore also a high cost. Containers are managed by a container engine that runs on top of a host operating system. This engine will create a new container for every application. The advantage is that it does not takes a lot of memory to run them, making them much faster. The containers will share the kernel as well as the resources, meaning that they will have a lower isolation level when compared to the hypervisor-type virtualization.

When using the same operating system either in a virtual machine or in a containerized system, we can see that the memory, CPU, and storage usage of the container is less than 10% of that of the virtual machine [10]. It will however come at a certain cost: due to the lower isolation and the difference with the virtual machine, we will not be able to reproduce the same scenarios and vulnerabilities inside the containerized system. The experimental results measuring the reproducibility of vulnerabilities can be found in [10], where a very high similarity has been found between the containerized system and the virtual machine.

The second range, KYPO CRP, is based on a set of state-of-the-art technologies such as containerization, infrastructures as code, microservices, and open-source software. It is not built on top of containerization but allows us to use them inside of the network we will try to simulate.

We will now explain more in detail the two solutions that exist in order to build cyber ranges on top of novel technologies.

CyExec*

CyExec* is a cyber range solution that is built on top of Docker, which is a popular solution to work with containers. With Docker, it is easy to spin up a set of containers. This can be

done by using a set of files containing the configurations of every container, called a Dockerfile. Then, by using Docker-compose, it is possible to take all of the single configuration files and link them together to create a network environment with potential vulnerabilities.

One particularity of CyExec* is that it adds scenario randomization. To do so, we will define a scenario that has multiple possible execution paths. This can be seen in Figure 2.3. For this, we must start by setting the different milestones that will be present in every scenario we will define. Then, we can consider every passage from one milestone to the next one as a mini-scenario. By taking one mini-scenario at random between every milestone, CyExec* can create a great variety of scenarios at random.

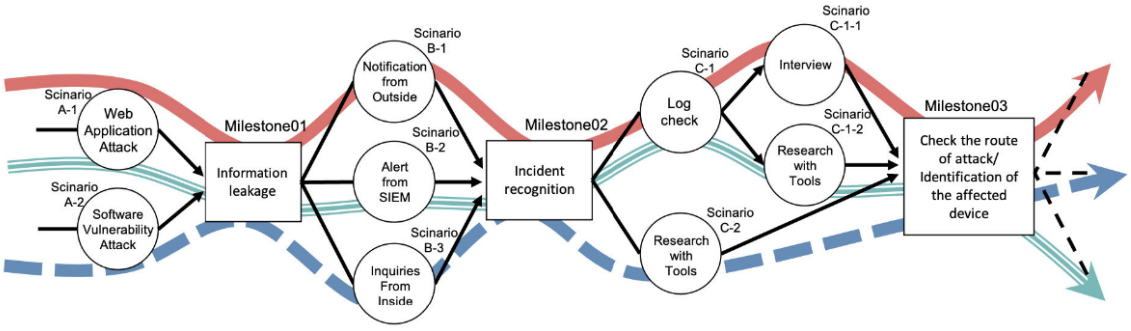


Figure 2.3: Example of a randomized cyber range scenario [10]

It can be seen that this product is very promising due to its low latency and memory usage. It must be noted, however, that at the moment, many features are limited when compared to the cyber range products that exist on the market.

KYPO CRP

The KYPO Cyber Range Platform (CRP) is built using a modular approach. To achieve this, it uses a cloud environment. This will allow it to be highly scalable, flexible, and cost-effective. The objective is for it to be able to run on top of different cloud computing platforms, such as OpenNebula[1] or OpenStack[2] [21]. At the moment, however, only OpenStack is supported.

The platform satisfies the following requirements:

- **Flexibility** – The networks that are simulated are of arbitrary topologies;
- **Scalability** – The platform scales well with the increasing number of users;
- **Isolation vs. Interoperability** – The platform can be used in complete isolation, where there is no communication between the different users and networks. It can also be used in a way where this communication is possible;
- **Cost-Effectiveness** – Low maintenance cost and it is deployable on commercial off-the-shelf hardware;
- **Built-In Monitoring** – Access to real-time and post-mortem monitoring data;
- **Easy Access** – Can be accessed using various means (browser, SSH, ...) and by any user, regardless of their experience level;
- **Service-Based Access** – Transparent access to the platform in the form of a service;

[1] `https://opennebula.io/`
[2] `https://www.openstack.org/`

- **Open Source** – Using open source projects (when possible) and is itself open source.

Most of these objectives can already be assured by the use of cloud computing platforms. Furthermore, the architecture that is used to achieve these requirements is shown in Figure 2.4

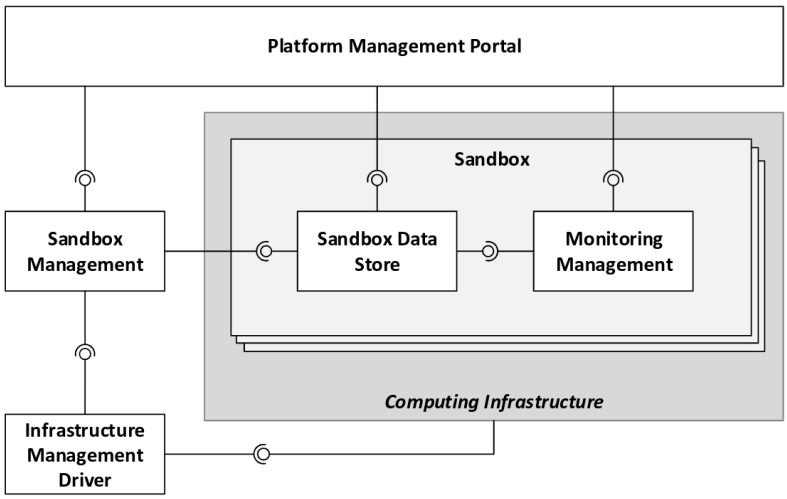


Figure 2.4: KYPO platform high-level architecture overview [21]

The main component that is used by KYPO is sandboxing. Those sandboxes can be either individual, shared between users that can use them independently, or even shared between different teams. As explained above, users will access their sandbox with a graphical web interface. There are however other components that come into play to ensure the good functioning of the entire system:

- **Infrastructure Management Driver** – This driver allows communication with the low-level computing infrastructure. KYPO is configured to work with OpenStack at the moment;
- **Sandbox Management** – Creates and controls the different sandboxes;
- **Sandbox Data Store** – Holds the information relative to the topology of a sandbox. It contains information about the end nodes, IP addresses, networks,...;
- **Monitoring Management** – Provides control over the monitoring of the infrastructure;
- **Platform Management Portal** – This module allows interaction from the user with the cyber range system. To this end, it provides a graphical interface with all the visual tools necessary.

2.4 Federated Cyber Range

As seen in the previous sections, when comparing the different classification methods of cyber ranges, we can classify them based on their type. The three main types that can be distinguished are public, private and federated. Since it is part of the goal of this work, we will now take a closer look at the federated ranges.

Every cyber range has its own complexity, uses a specific technology and is specialized in a certain domain. In some cases, we want to access a service that is too specialized in

order to implement it in the cyber range environment we are developing. This is where the federation of ranges comes into play. By doing so, we will inter-connect multiple cyber ranges, each range having its own specialty and location. We could for example want to connect to a cyber range containing a set of SCADA[3] (Supervisory control and data acquisition) systems, one containing a set of medical records, etc. When connecting the various ranges together, it becomes trivial to access the different services that are located at another place.

There are different ways of interconnecting such cyber ranges, and we will explain some of the most common ways to do so.

A first, and very naive way for making the connection, is simply to route the traffic from a router on the first network to a router on the second network. In doing so, we would achieve our goal of cyber range federation. This is however, not the way to do, since all the traffic flowing between the two ranges will be unencrypted, and therefore visible for any malicious user listening on that connection.

By solving this problem, we come to the second solution: using a VPN. VPN stands for Virtual Private Network, and is a way of inter-connecting two networks in a secure way over an insecure channel (here this is the internet). A VPN achieves this by creating a VPN tunnel, where all the data that passes through this tunnel is encrypted.

There exist multiple types of VPN connection, either based on their utilization or on the OSI layer at which they operate:

- Utilization:
  - Host-to-Site - In this case, a single host needs to access an entire network that is at a remote location. A VPN connection can therefore allow that single host to access the remote network in a secure way;
  - Site-to-Site - In this case, two entire networks that are separated by the internet, will be connected together by the use of a VPN connection.
- OSI Layer:
  - Data link layer (Layer 2) - This is the most expensive type of communication. Normally, the protocols that are used over this layer do not add encryption [7]. This means that it needs to be added manually;
  - Network layer (Layer 3) - This type of VPN will be able to encrypt any traffic;
  - Application layer (Layer 7) - This type of VPN uses the SSL/TLS protocol in order to communicate in a secure way.

There are many protocols that exist already, that operate at various OSI layers and permit different types of utilization:

- OpenVPN - Open-source VPN that operates over SSL/TLS. It allows encryption of data and authentication using several methods.

[3]https://en.wikipedia.org/wiki/SCADA

- IPSEC - Set of protocols that operates at layer 3 of the OSI model in order to tunnel encrypted data. It also allows to authenticate the connection. An in depth review of the working of IPSEC can be found in [1];
- IKEv2 - This is not a VPN technology as such, but it is a protocol (used by IPSEC) that allows secure key exchange over the internet.
- Wireguard - More recent VPN solution. It transmits encrypted data over UDP and operates at the layer 3 of the OSI layer.

Due to the emerging need of federation of cyber ranges, there are also some frameworks that are being developed. One of such solution, which is also open-source is the ECHO Federated Cyber Range (E-FCR) [13]. The goal of E-FCR is to be a marketplace for cyber range services in Europe. It provides a user-friendly portal in order to be able to access the multiple cyber ranges that are connected to is. That portal will also distribute and configure a given scenario on a pool of interconnected cyber ranges. In that way, it is directly possible to know if a certain scenario can be implemented with the cyber ranges that are connected to the E-FCR.

2.5 Choice of technology

One very important criteria in order to choose the technology that will be used throughout this work, is for it to be open-source as much as possible. Doing so allows a high degree of possible customization. Furthermore, it needs to be actively maintained and allow a broad set of different scenarios. Another criteria that is important is the installation difficulty.

For all of these reasons, we have chosen KYPO CRP which is open source and most of the tools that it uses are open-source as well. It uses state-of-the-art technology and is built on top of an OpenStack cloud computing platform. The installation steps can be found on their web page[4] and are very easy to follow. Furthermore, an element that made the choice decisive, is that KYPO offers the possibility to install a Lite version of the cyber range platform. Making it very easy to implement a small scenario, when using less resources than if we were to spin up the full version.

With regards to the VPN solution, it is OpenVPN that has been chosen. This choice has been made because of its easy installation and setup process. Furthermore, this VPN solution has proven to work well with networks that are behind NAT (Network Address Translation). We will see further in this work that the routers we want to connect to inside the cyber range platform, could be hidden behind multiple layers of NAT.

2.6 Summary

From this chapter, we can conclude that there are a lot of different cyber ranges that exist. Some of them are publicly available, others are private. The overall architecture of such systems is however the same: they will try to create a model of the target infrastructure based on different virtualization methods. Those methods can vary based on the goal that

[4]https://docs.crp.kypo.muni.cz/installation-guide/installation-guide-overview/

wants to be achieved with the cyber range.

Furthermore, there are a lot of different ways to classify cyber ranges. Different studies use different criteria in order to do so. In [23], the criteria that are used are very detailed. Those aspects are scenarios, monitoring, teaming, scoring, and management. In a different manner, [19] rather analyzes the application domain, the type, the team formations that are possible, the experimentation methods,...

Due to the extensive documentation inside existing articles about a comparison of the state-of-the-art, a summary of the described papers has been made. After that, the choice has been made to put the focus on two novel implementations of cyber ranges: CyExec* and KYPO CRP. The first solution relies solely on containerization as its virtualization layer and the latter uses a combination of hypervisor-based technologies, container-based solutions, microservices,... Cyber range platforms that rely only on containers are still in the early stage of development and, as has been described with CyExec*, they are still behind the more classical hypervisor-based solutions. This is why KYPO CRP uses a combination of various technologies to try and get the best out of every technology.

Finally, there is still a lot of research going on in the field of cyber ranges. Since it is still a new technology, researchers are trying to apply the concept of a cyber range in various domains: [6] studies the use of AI to create scenarios, [12] explores the possibility of a cyber range for IoT devices, ... Lots of work are also in progress to generate realistic background traffic inside such cyber ranges, which would make the training on such platforms closer to reality.

the management of cybersecurity, many roles play

Chapter 3
Objectives and Implementation

In this chapter we will start by summarizing the different objectives of the system that we are trying to build. After that, we will go more in depth in the implementation of the different steps that will be described inside the objectives section.

3.1 Objectives

The objective of this Master's thesis is to implement, and automate as much as possible, a way to add federation inside of a scenario when using KYPO Cyber Range Platform. As has been stated in previous chapter, we would like to be able to connect to to remote services or even entire networks. This is needed when we would like to use highly specialized tools that are already implemented in some sort of cyber range environment. When this environment is located at a distant location, we would like to be able to connect to it in a way that is as transparent as possible for the user of the local cyber range.

In order for this setup to be as easy as possible to use, we will use tools that have been presented in the previous chapter: a cyber range platform (KYPO CRP) as well as a VPN solution (OpenVPN). We will first need to understand how the different scenarios are created inside KYPO, as well as see if it is possible to automate such configuration. Then, it is important to understand how to configure correctly the OpenVPN setup.

3.2 Requirements

Since KYPO Lite will need to spin up an entire OpenStack Cloud computing instance, there are some requirements that need to be met on the hardware to use. The most limiting component is the RAM (Random Access Memory), which is much solicited when performing virtualization. The authors of the KYPO project advise to use 32GB of RAM and 4 virtual CPUs at least in order for KYPO Lite to work. The recommended amount is however 44GB and 8 virtual CPUs. These values can be configured at the installation of the cyber range solution. The installation of KYPO Lite is very simple as it only consists in running a Vagrant script that is on the Gitlab repository[1].

The hardware specifications of the computer on which KYPO lite has been installed are:

- 48GB of DDR4 RAM
- Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz with 6 cores
- NVIDIA graphics card: GeForce GTX 1060 6GB

[1] https://gitlab.ics.muni.cz/muni-kypo-crp/devops/kypo-lite

3.2.1 Troubleshooting

It is possible that, when launching the vagrant file, you get some errors in the last steps such as 'TLS handshake timeout error'. To fix this, it is needed to comment the last few lines that call 'terraform init' twice inside the Vagrantfile provided on the Gitlab of KYPO Lite. Then, after running the vagrant file without these lines, ssh into the virtual machine with the command *vagrant ssh – -t 'sudo su'* and execute the commands that have been commented out manually. This should fix the issue.

Secondly, once the machine is spun up, you can halt and resume it with the commands 'vagrant halt' and 'vagrant up'. Once that the machine has been rebooted, all OpenStack instances will be in **SHUTOFF** state. It is necessary to activate them all. This can be done in a one-line command, again after having connected to the virtual machine via ssh, which is:

```
$ openstack server list --status SHUTOFF -f value -c ID |
    xargs -n 1 openstack server start
```

After some time, the different virtual machines will be set up and the system will be usable again.

3.3 KYPO Architecture

In order to be able to connect multiple cyber range scenarios together, it is necessary to understand the KYPO architecture. More importantly, since we are using the Lite version of KYPO, we need to understand the different layers that need to be crossed before being able to reach the internet.

When analyzing more in detail the different configuration files for KYPO and the state of the different virtual machines after having setup multiple sandbox instances, we can show an overview of KYPO's architecture in Figure 3.1.

When we are installing the Lite version of KYPO, it will install a local OpenStack instance on top of which it will build all the different virtual machines machines. We will now explain briefly how KYPO sets up its different internal networks in order to create the sandboxes. The steps described below are only a subset of the operations performed by KYPO. The whole set of them will not be described here, as it is not the purpose of this work. They can however be recovered by analyzing the different configuration files on the Gitlab repository of the Masaryk University[2].

The different operations are:

1. Create a 10.1.2.0/24 network that is shared with the computer in order to build the OpenStack instance on top of it;

2. Install OpenStack on top of the created network;

[2]https://gitlab.ics.muni.cz/muni-kypo-crp

3. Add an OpenStack router that has the ability to communicate with the external world. This can be done in OpenStack by setting one of the interfaces as being part of an external network. The other interface is part of the internal network. In the case that will be used throughout the rest of this work, the interface facing the external network has IP address **10.1.2.192**. The interface facing the internal network has an IP of **192.168.64.1** and is connected to the **192.168.64.0/18** network.

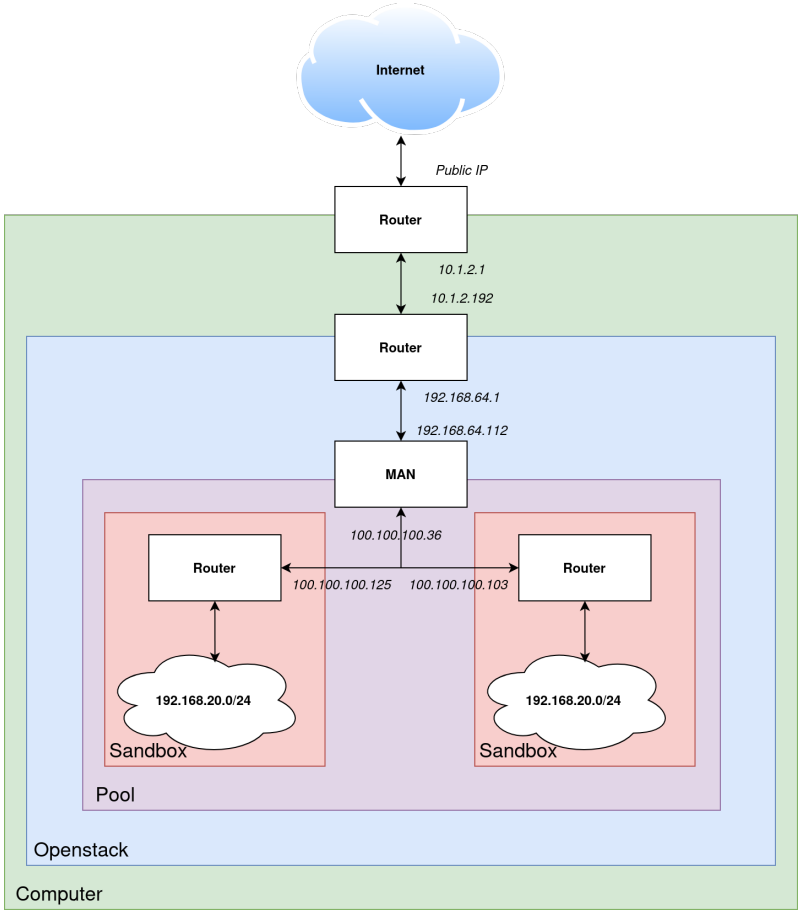


Figure 3.1: Overview of routing between sandbox and internet

At this point, we have set up the green and blue areas that are shown on the figure above. The rest of the network shown in the diagram will be setup when we create a scenario. As an example, we will add two instances of the same scenario in which we have a simple layout: one router connected to the 192.168.20.0/24 network.

In KYPO, as has been described in Chapter 2, every scenario will run inside its own sandbox. If we need to spin up multiple instances of the same scenario, we need to create a pool that will contain multiple copies of the same sandbox. To come back to our example, we will have the following steps:

1. Create a virtual machine inside the 192.168.64.0/18 network with a random IP address. This will be the management node, which will be common to every sandbox inside the pool;

2. Create an interface on the MAN node and assign a random IP address in the 100.100.100.0/24 range to it;

3. Create 2 instances of a 192.168.20.0/24 network with the different virtual machines as is specified in the configuration of the scenario;

4. Add an interface connected to the 100.100.100.0/24 network on every router and assign a random IP address to them.

It has to be noted that this is a very simplified view of what is happening in reality. For example, KYPO also sets up a 192.168.128.0/17 network where the MAN node is connected to every single virtual machine that is created inside the pool for management purposes,...

We can now understand how the purple rectangle, that represents a pool, and the red ones, representing sandboxes, have been created. It is now easy to extend this to the creation of multiple pools. In that case we will simply replicate the purple rectangle from the figure above on a newly created MAN node.

A last element that needs to be added in order to have the full picture about the routing from and to a sandbox inside KYPO, is the use of address translation. A first place where this happens is on the OpenStack router that is located at the border between the green and blue rectangles. At that location, there is source NAT that is enabled. This means that there will be an address translation on the source address when the packet is leaving the network: the source IP will be 10.1.2.192 and the source port will be a random port that the router will keep in memory in order to be able to do the translation on the other way around as well.

A second place where there is a NAT is on the gateway to access the internet, both on the computer and on the router connected to the Internet Service Provider.

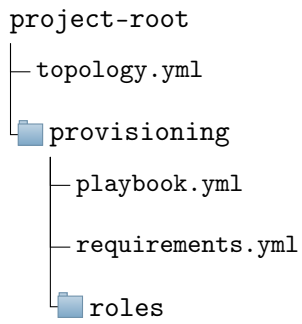
We can see that we have multiple layers of "natting", confirming the statement that has been written in previous chapter about the choice of the VPN solution. Since OpenVPN has a reputation of being very NAT-friendly, it has been chosen as it would fit very well in our situation.

3.4 Sandbox Definition

In KYPO, a sandbox definition represents a snapshot of a network topology with a given state. It is possible to also add a training definition on top of this in order to create our scenario. We will however only talk about the sandbox definition part, because we do not need to specify a training in order to be able to test our implementation.

The sandbox definition that is implemented in KYPO is not based on a graphical interface, but rather on configuration files that can then be reused and modified. The configuration is based on the Ansible playbooks, that enables to configure machines from a central location based on a set of given configuration files. In order to upload a configuration to KYPO, it is needed that it is located inside of a git repository. This means that we need to create one repository for every sandbox definition we wish to create.

The overall structure of the git repository that is needed in order to provide a sandbox definition is the following:



It is important to keep this directory and file structure with the same names, otherwise, the KYPO platform will not be able to recognize the different configurations. It will therefore create an error stating that the structure is not correct. We will briefly explain the utility of every file and folder that is shown above:

- topology.yml - File containing the topology that will be used inside the sandbox. This is where we declare the networks that we will use, the type of every virtual machine, the routers and the connections between all of the elements we just cited.
- provisioning - Folder that will contain the information that is relevant in order to provision the different machines that have been declared inside the topology file.
  - playbook.yml - The root Ansible playbook that defines the roles for every host. It is of good practice to not perform every action in this playbook file, but rather define different roles. Every role will have a specific function and be located inside the roles folder. The syntax and functions used inside this playbook file is the same as for a classical Ansible playbook and we will therefore not go more in depth into this subject. We will explain the functions we use later, as we encounter them in our implementation.

    It has to be noted that there are some additional groups that can be used on the top of the classical Ansible Host Groups. These allow for example to address all hidden hosts, all user-accessible hosts or even all routers. Additional details on the provisioning of a sandbox in KYPO can be found back in the documentation[3].
  - requirements.yml - File containing references to roles that might be located on other git repositories.
  - roles - Folder containing the different roles that we have created. Inside this folder, we will have one new folder for every role that has been created. The folder of the role will have the name that will be used in order to use the role itself.

Further in this work, we will use the example configuration that is provided by KYPO[4] as a starting point. In these configuration files, a simple scenario is created. We have one

[3] https://docs.crp.kypo.muni.cz/user-guide-advanced/sandboxes/sandbox-provisioning/
[4] https://gitlab.ics.muni.cz/muni-kypo-crp/prototypes-and-examples/sandbox-definitions/kypo-crp-demo-training

router which is connected to two networks: 192.168.20.0/24 and 192.168.30.0/24. On the first network we have a client at IP 192.168.30.5, and on the second network we have a vulnerable server with IP 192.168.20.5.

3.5 OpenVPN

Now that we know how to configure a sandbox, we can start working on the federation of different cyber ranges. As explained before, we will use OpenVPN in order to achieve this. In KYPO, the routers are configured as Linux machines running Debian in order to have a maximum level of configurability. Since every router is directly connected to the MAN node, which will route the traffic to the internet, it seems to be the ideal place in order to put the VPN server or client. In that way, all the traffic from the network that will try to go to an IP range that is not defined inside the network, will by definition pass through that router. The router can then decide to either forward that to the internet, or, if a VPN client or server is configured and the target IP address matches the network on the other side of the tunnel, pass the traffic through the VPN tunnel that has been created.

There are two main ways to apply encryption on an OpenVPN tunnel: either using asymmetric keys or pre-shared symmetric keys. The first case implies the creation of a public key infrastructure (PKI) and the use of signed certificates. This method is useful when the two parties do not know each other in advance. In the second case, we assume that the two parties have exchanged a cryptographically secure symmetric key in advance. They can then both use that key and pass it to OpenVPN.

Since OpenVPN can only create an encrypted tunnel between two hosts, we need to tackle the case when there are more than two cyber ranges that need to be connected together. For this there are two main solutions. A first solution is to have a central federated gateway as proposed by [14]. In this architecture, every cyber range that wishes to join the network will create one new connection to the central gateway. Once this connection is made, it will be part of the federated network and therefore also able to access all the resources that are at its disposal.

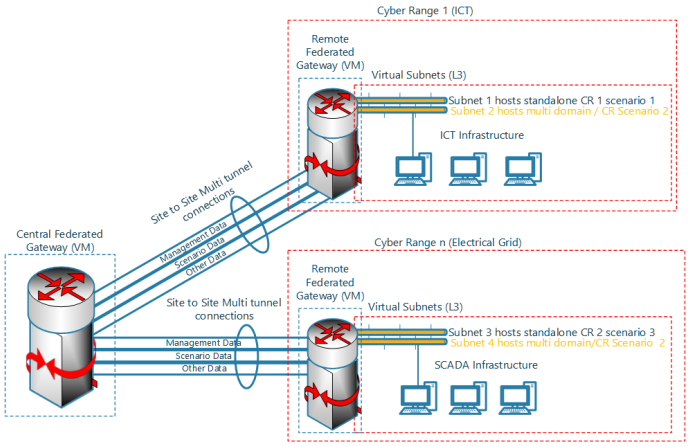


Figure 3.2: VPN Interconnection as proposed by [14]

A second solution is to use the peer-to-peer model, where every cyber range will have a VPN tunnel with every other cyber range that is connected to the federated network.

Both these solutions have both drawback and advantages:

- Central gateway solution:
  - + Every range sets up one connection only;
  - − Single point of failure of the central gateway;
  - − Need for a centrally managed server for the central gateway.
- Peer-to-peer solution:
  - + Distributed management. Even if one range fails, all other ranges can still communicate together;
  - − Every cyber range needs to set and maintain up N-1 connections, where N is the total number of cyber ranges inside the federated network;

In practice, except for the need of an additional server to play the role of the central gateway, the implementation of both solutions does not differ by much. The main differences are the setting up of the different tunnels and the site that is responsible for the routing. We will therefore choose the peer-to-peer method to implement inside KYPO. This can however easily be adapted in order to be able to work with a central gateway as well.

3.6 Manual installation

We will first explain how to install the tunnel manually and, after that, transfer this installation steps to the Ansible playbooks that define a sandbox inside of KYPO.

The first step is to install OpenVPN on the different machines. The routers are using Debian Stretch, however if we try to update the list of available updates with '*sudo apt update*', we will get a set of errors:

```
E: The repository 'http://deb.debian.org/debian stretch-updates Release'
    does not have a Release file.
N: Updating from such a repository can't be done securely,
    and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation
    and user configuration details.
E: The repository 'http://security.debian.org stretch/updates Release'
    does not have a Release file.
N: Updating from such a repository can't be done securely,
    and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation
    and user configuration details.
```

This is because the Stretch version of Debian is not the latest anymore and we must take the archive repositories of Debian. To fix this, we need to change the content of the file '*/etc/apt/sources.list*' to:

```
deb http://archive.debian.org/debian/ stretch main contrib non-free
deb http://archive.debian.org/debian-security/ stretch/updates main contrib non-free
```

After saving the changes, we can now install OpenVPN by first updating the list of applications with '*sudo apt update*' and then install OpenVPN with the command '*sudo apt install openvpn*'.

Once OpenVPN is installed, a folder at '*/etc/openvpn*' is created. Every file inside that folder that has a '*.conf*' extension will then be treated as an OpenVPN configuration file. We will then create a file at that location called '*connection1.conf*'. The content of the file for both the server and client are very similar. For the server, we will use the following template:

```
1   port <Port-Nr>
2   proto udp
3   dev tun
4   ifconfig 172.0.0.1 172.0.0.2
5   cipher AES-256-CBC
6   secret secret.key
7
8   comp-lzo
9
10  keepalive 10 60
11  ping-timer-rem
12  persist-tun
13  persist-key
```

The explanation for the main lines are:

- *l1* – Specifies the port on which the OpenVPN server will listen;
- *l2* – Specifies the protocol that is used during the communication;
- *l3* – Name of virtual network interface;
- *l4* – Specifies the IP addresses of the local and remote tunneling interface (in that order);
- *l5-6* – Specifies the name of the file containing the secret key as well as the cipher suite to use;
- *l8* – Tells OpenVPN to compress the data before sending it to save bandwidth;
- *l10-13* – Makes OpenVPN more resistant to potential failures

In order to configure the client, we will use the same configuration file, except that we need to replace the first line of the file by the following line:

```
remote <OpenVPN-Server-IP> <Port-Nr>
```

The next element that is missing, is the generation of the file containing the secret key. Since this file will need to be shared between the VPN client and server, we need to make sure to copy it from one instance to the other. OpenVPN provides a command-line tool that will generate this secret key. The command to use is the following:

```
$ openvpn --genkey secret secret.key
```

Finally, we need to make sure that the IP forwarding is enabled on the machine that will forward the traffic to an entire network. This could be the case on the server side, but also on the client side. The way to achieve this on a Linux machine is by using following command:

```
$ sudo sysctl -w net.ipv4.ip_forward=1
```

If it is the case that we need to route the traffic to a specific sub-net through the tunnel, we can add it to the configuration file as well. This allows us to not modify the Linux internal routing table manually. The line to add to the file is the following:

```
route <Remote-Subnet> <Remote-Network-NetMask>
```

As an example, if the network that is located at the server's location has a netmask of 192.160.2.0/24, we would add the line '*route 192.160.2.0 255.255.255.0*' to the client's configuration file. By doing so, any packet with a destination that matches that subnet and passes through the VPN client will automatically be routed through the VPN tunnel.

Once that the different files have been generated, they can be placed inside the '*/etc/openvpn*' folder. In order to start the VPN server, we can run both of the following commands:

```
$ sudo systemctl enable openvpn@connection1.service
$ sudo systemctl start openvpn@connection1.service
```

This will start a service in the background that will listen for any incoming connection. We can check if the command was successful by looking at the output of two commands. The first command is to see if the service has been started correctly:

```
$ sudo systemctl status openvpn@connection1.service
```

If the status is set as enables, then it means that the service has been created successfully. The second command is to see if there is a tunneling interface that has been created:

```
$ ip link show
```

If there is an interface starting with 'tun' in the list of interfaces, then we can conclude that OpenVPN was launched successfully.

3.7 Process Automation

Now that we know how to setup an OpenVPN tunnel manually, we can start writing the automation scripts that are needed by KYPO for the sandbox definition. There are multiple steps that need to be taken and they will be discussed further in this work.

Preparation

There is one step that must be executed in order to prepare the router for the installation of KYPO. As explained in previous section, we need to update the '*sources.list*' file in order to be able to install any packages on the router. In order for the code to be as modular as possible, we will create a separate role for that called '*debian9-update-sources*'. The only purpose of this role will be to replace the content of the file located at '/etc/apt/sources.list'. The folder structure of this role can be seen below.

```
roles
└── debian9-update-sources
    ├── tasks
    │   └── main.yml
    └── files
        └── sources.list
```

Let us start with the content of the main.yml file. It contains only one task, and is using the Ansible copy module to copy a file to the target machine. By default, Ansible will look for files inside the '*files*' folder, which is why we added the file to copy inside that folder. The content of the main file is:

```
- name: Update sources.list file
  copy:
    src: sources.list
    dest: /etc/apt/sources.list
    owner: root
    mode: 0644

- name: Update cache
  apt:
    update_cache: yes
```

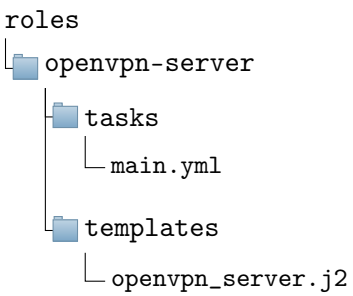
These two tasks will first copy the file '*sources.list*' to '*/etc/apt/sources.list*', set its owner to root and set the appropriate permissions. After that the cache will be updated. The content of the '*sources.list*' file is, as has been shown in the previous section:

```
deb http://archive.debian.org/debian/ stretch main contrib non-free
deb http://archive.debian.org/debian-security/ stretch/updates main contrib non-free
```

Whenever we will want to install a package on a machine that is running Debian 9, we will need to add this role first to make sure that the cache is updated properly.

Automating server creation

We will first start by automating the generation of an OpenVPN server using KYPO. For this we will create a role called '*openvpn-server*', meaning we create a folder with that name inside of the '*roles*' folder. The main file that will be used for that role must be called '*main.yml*' and located into a sub-folder called '*tasks*'. The folder structure that is used can be seen below. We will explain the content of each file as we go through the content of the main file.



The first step is to install OpenVPN and enable IPv4 traffic forwarding. These two steps can be done by using the *apt* and *sysctl* Ansible modules respectively. When translating this to the Ansible format, we obtain:

```
- name: Install Openvpn
  apt:
    pkg:
    - openvpn
    update_cache: yes

- name: Enable IPv4 traffic forwarding
  sysctl:
```

```
    name: net.ipv4.ip_forward
    value: 1
```

The next step is to handle the different secret keys that will be needed in order to setup the connections. Due to the fact that we want the role to be as modular as possible, we give the possibility to create multiple OpenVPN instances. The task copying the keys to the router will therefore loop over a set of entries it has received, and copy all of them inside a 'secrets' directory. The code for this part of the script is the following:

```
- name: Create directory for secret keys
  file:
    path: "/etc/openvpn/secrets"
    state: directory
    owner: root
    mode: 0600

- name: Copy secret key to server
  copy:
    src: "{{item.secret_file}}"
    dest: "/etc/openvpn/secrets/{{item.name}}-secret.key"
    owner: root
    mode: 0400
  with_items: "{{ openvpn_instances }}"
```

We can observe that we first create a '*secrets*' directory that is owned by root and only readable and writable by the root user. Then, we loop over all the OpenVPN instances with the 'with_item' keyword. By doing so, every instance will be accessible by using the 'item' keyword. We have decided that every instance is a dictionary that contains at least a name and a secret_file key. More keys will be added as we go through the next tasks.

The following task is to create an OpenVPN configuration file for every instance. We will again loop over all instances, but instead of using the '*copy*' module, we will use the 'template' module. Templates are files that use the Jinja2 templating language and allow to copy files, while inserting some variables inside of it. This means that we will not have a static file anymore, but a file that will contain variable information. The file that is used must be inside the '*templates*' directory and is called '*openvpn_ config.j2*'. The content of the file is very similar to the sample configuration file that has been proposed in the previous section:

```
1   port {{item.server_port}}
2   proto udp
3   dev tun
4   ifconfig {{item.tun_ip_local}} {{item.tun_ip_remote}}
5   cipher AES-256-CBC
6   secret /etc/openvpn/secrets/{{item.name}}-secret.key
7
8   {% for item in item.extra_options %}
9   {{ item }}
10  {% endfor %}
11
12  comp-lzo
13
14  keepalive 10 60
15  ping-timer-rem
16  persist-tun
17  persist-key
```

The way to copy this template to the router while filling in the relevant information is the following:

```
- name: Copy openvpn configuration file
  template:
    src: openvpn_config.j2
    dest: /etc/openvpn/{{item.name}}.conf
    owner: root
    mode: '0644'
  with_items: "{{ openvpn_instances }}"
```

All the information that would change if we created a different configuration has been put inside variables. We now can have a clearer picture of the content of the dictionary that describes an OpenVPN instance. It must have following elements:

- name – Unique name that will serve as an identifier for the connection
- secret_file – Path to the file containing the shared secret key
- server_port – Port on which the server will listen for incoming connections
- tun_ip_local – IP address that the local tunneling interface will have
- tun_ip_remote – IP address that the remote tunneling interface will have
- extra_options – If there are additional entries that need to be inserted inside the configuration file, they can be put inside this list

Now that the configuration files and the corresponding secret keys are in place, we can start the OpenVPN instances. This can be achieved by using the Ansible service module:

```
- name: Enable services
  service:
    name: "openvpn@{{item.name}}.service"
    enabled: true
  with_items: "{{ openvpn_instances }}"

- name: Start openvpn
  service:
    name: "openvpn@{{item.name}}.service"
    state: started
  with_items: "{{ openvpn_instances }}"
```

Automating client creation

The creation of the client is very similar to the creation of the server. In fact, the files containing the tasks is the same. Similarly to the manual installation, it is the content of the configuration file that will need to be adapted. Due to this very small change only, we will not create a separate role, but rather adapt the role that was created for the server. There are two modifications that are necessary in order to be able to generate either a client or a server using the same role:

1. Add a boolean variable called '*server*' that will indicate if the current instance is a server or not. If not, then it is a client by definition.

2. Adapt the jinja2 template file in order to take this modification into account. For this, we need to replace the first line of the template that was presented above. The configuration becomes:

```
{% if item.server %}
    port {{item.server_port}}
{% else %}
    remote {{item.remote_ip}} {{item.server_port}}
{% endif %}

proto udp
dev tun
ifconfig {{item.tun_ip_local}} {{item.tun_ip_remote}}
cipher AES-256-CBC
secret /etc/openvpn/secrets/{{item.name}}-secret.key

{% for item in item.extra_options %}
{{ item }}
{% endfor %}

comp-lzo

keepalive 10 60
ping-timer-rem
persist-tun
persist-key
```

Listing 1: Jinja template of the configuration file that can be used for both a client and a server

By performing these two simple modifications, we now have a single role that can install either a client or a server. The name of the role is not fitting anymore, and will therefore be changed to simply '*openvpn*'.

Further automation

Now that we have a completed script that allows us to setup either a client or a server inside a KYPO sandbox, we could ask ourselves if it is not possible to further automate the process? In our case, we only had to inter-connect two different instances. We can however see that the variables that are used for the *openvpn*, as can be seen in Listing 3 role, can quickly grow in size. This will be true especially when we will want to connect more than two instance together. When we want to connect N instances in a federation, we would have to start $N-1$ connections on every individual instance. Furthermore, there are some values, such as the tunneling ip addresses and port numbers that could be automatically chosen. The secret keys as well could be automatically generated for every connection.

We will try to achieve this by creating a Python script. In order to do so, we must first create a way to specify the topology of the different instances inside the federation. This representation will be the main input to the python script. This document will be implemented using the JSON syntax. A high-level specification of this schematic can be given by the following code:

```
{
    "type": "object",
    "properties": {
        "<unique_name>": {
            "type": "object",
            "properties": {
                "ip": {
                    "type": "string",
                    "required": true
                },
                "internal_ip": {
                    "type": "string"
                },
                "tun_ip": {
                    "type": "string",
                 "default": "172.0.0.1/24"
                },
                "kypo": {
                    "type": "boolean",
                    "default": true
                },
                "conn_to": {
                    "type": "array",
                    "items": { "type": "string" }
                }
            }
        }
    }
}
```

Listing 2: Schema of the topology description used by the automation script

The JSON file represents an object where every entry describes one cyber range instance. The key of the entry is a unique name representing the range and the value associated to that key has the following elements:

- "ip": A string representing the IP address associated with the unique name. This field is mandatory.
- "internal_ip": A string representing the internal IP network that will be reachable by the other ranges inside the network. This field is optional;
- "tun_ip": A string representing the range of IP addresses that we want to assign to the local tunnelling interface. This field is optional, and, if omitted, will default to the '172.0.0.1/24' network;
- "kypo": A boolean value indicating whether the current instance is a KYPO instance or not. This field is optional and defaults to 'true';
- "conn_to": An array of strings representing the unique names of other dictionaries to which a connection is established. This field is optional. If it is not present, it will be equal to a list of all of the other ranges that are present inside this configuration file. It allows to have some cyber ranges that are connected to only a subset of the ranges present inside the federation.

Once the specification for the input configuration file have been set, it is possible to design the script that will transform this to usable configurations. The entire code will not be shown here, it can however be found back on a GitLab repository located at `https://gitlab.com/mat0110/kypo-topology/-/tree/main/python`. In order to run this code there is one dependency that must be installed, Jinja2, which can be done by running the command: 'pip install Jinja2'.

We will now break down how the transition from the configuration file, as specified in 2, to a setup that is usable by KYPO is done.

First, the file is read by Python and parsed to a dictionary. While doing that, we check if the "conn_to" field is empty or not. If it is, then we generate a list of all of the other members and add it to this entry. This step is needed in order to facilitate the verification step, where we will check that every entry in that list actually corresponds to a valid entry inside the input configuration. After this is done, we perform the actual validation of the file in order to make sure that it meets the specification.

At this stage, we can start iterating over the entries and generating the different elements. We will need, for every instance, to iterate over the list of ranges it is connected to (found inside the "conn_to" field). For every entry in this list, we need to find the values that will be assigned to the parameters that have been defined previously: name, secret_file, server_port, tun_ip_local, tun_ip_remote, extra_options and optionally also server_ip. These values are needed in order to generate a working openvpn role inside KYPO.

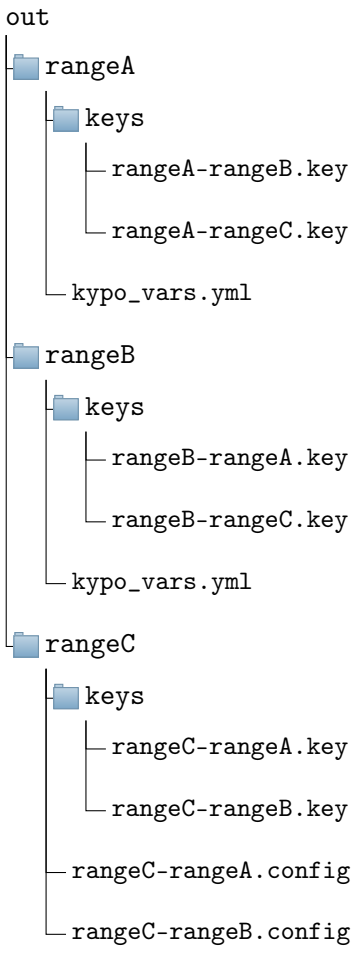
We have seen earlier in this work that we either have an openVPN server, or a client. The steps performed in both cases are not the same:

- OpenVPN server – This option is chosen if it is the first time we encounter the name of the cyber range inside the JSON configuration file. In this case we will follow the following steps:
  1. Generate a secret key;
  2. Generate a random port to listen on. It has been chosen to choose one between 10000 and 65535 in order to limit the chances of this port already being used;
  3. Generate a local tunneling IP address, chosen randomly inside the specified network;
  4. Generate a remote tunneling IP address, chosen randomly inside the specified network.
- OpenVPN client – This option is chosen if it the target cyber range has already been processed. In this case we will follow the steps:
  1. Get the configuration that was generated by the peer instance, which instantiates an OpenVPN server;
  2. Get the server ip address, which is the ip address of the peer we are connected to;
  3. Get the server port from the configuration of the peer instance;
  4. Get the local tunneling IP address as the remote IP address inside the configuration of the peer instance;
  5. Get the remote tunneling IP address as the local IP address inside the configuration of the peer instance;

6. Copy the secret key, in order for it to be the same one as the one that the server uses.

For every cyber range instance, we will create a separate directory. Inside this directory we will place the output configuration as well as all of the different secret keys that have been created. The last step is now to generate these output configuration files. We have two cases we need to consider:

- The cyber range we are generating the configuration for is using KYPO. In this case we output the list of connections with their corresponding variables to a file inside the directory. The content of this file will then simply have to be pasted inside the Ansible file that is used in order to provision a sandbox that is using the 'openvpn' role. The folder containing the keys will also need to be copied;
- The cyber range we are generating the configuration for is not using KYPO. In this case, we will generate the OpenVPN configuration file by using the same template as the one shown in Listing 1. This means that we will have one configuration file for every connection to, or from this range.



As an example to illustrate the output structure, we will imagine that we only have three ranges inside our federation: the first one is called "rangeA", the second one "rangeB" and the last one "rangeC". The two first ones are using KYPO, while the third one is not. The output structure will be can be seen on the left side. We can see that there are three directories that have been created, one for every cyber range. Inside every directory there is an additional directory containing the keys, and one, or multiple files defining the configuration depending on the type of cyber range.

Chapter 4
Results

In this chapter we will put in practice the different scripts that have been set up and test them on an instance of KYPO Lite. We will differentiate three different cases for the implementation:

1. The KYPO instance contains the VPN client. The VPN server is not using the KYPO cyber range solution;
2. The KYPO instance contains the VPN server. The VPN client is not using the KYPO cyber range solution;
3. Both ends of the VPN tunnel are running inside a KYPO instance.

It has to be noted that, technically, the OpenVPN client and server have the same capabilities. They only differ in the fact that it is the client that needs to initiate the connection towards the server. For the rest it is exactly the same. This means that the first and second case written above will be very similar.

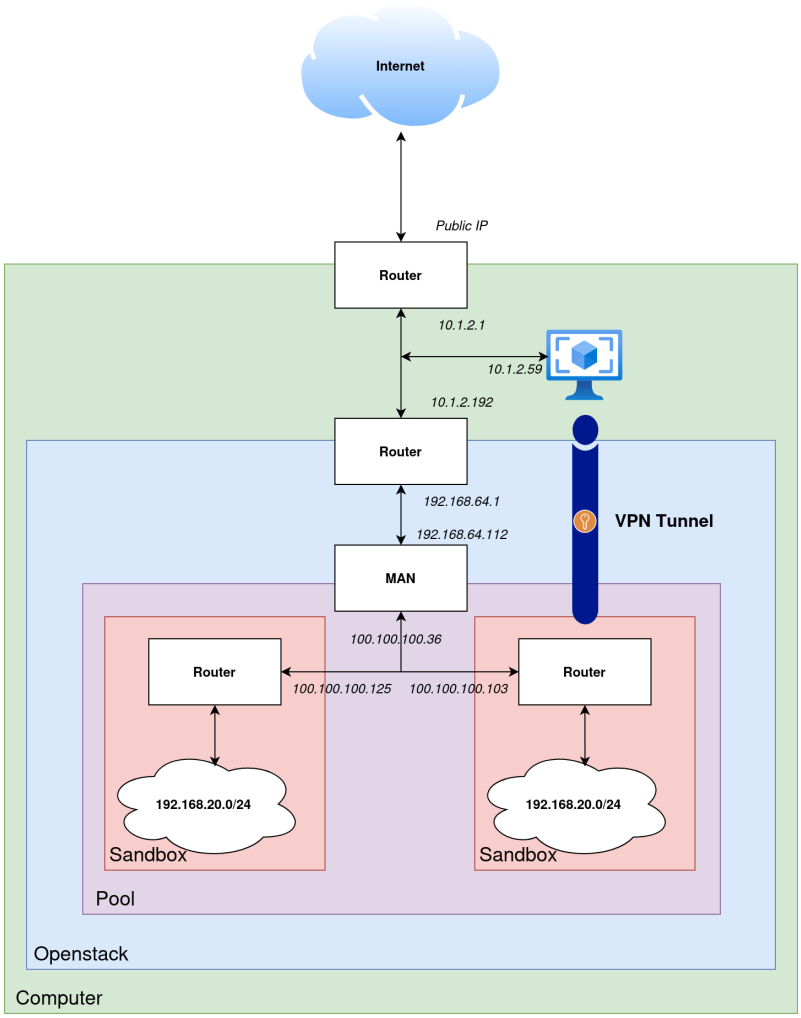


Figure 4.1: Overview of the routing between sandbox and internet with an additional virtual machine

For the testing of our configuration, we will add a virtual machine in our setup from Figure 3.1, which will be our external host to which we need to connect in a secure way. The update routing information can be seen in Figure 4.1. The virtual machine that has been added has one interface connected to the 10.1.2.0/24 network with IP 10.1.2.59 and another interface that is connected to the 192.168.2.0/24 with an IP address of 192.168.2.246. The second interface has been added for testing purposes only and there are no other machines connected to that sub-network.

In the case where the initial OpenVPN connection needs to go towards a sandbox inside the KYPO instance, there is a manual step that will need to be performed after the sandbox has been created. This will be the case in the second and third scenarios that have been depicted above. This step consists in adding a static route inside the OpenStack router. To recall, in the example shown in Figure 4.1, it is the router at the border between the green and blue rectangle. The router will need to forward any communication to the public IP of the router inside the sandbox to the MAN node responsible for that sandbox. In the example, we would need to forward any traffic to the IP 100.100.100.103 to the IP 192.168.64.112. This step can not be automated since both the IP address of the MAN node and of the router are randomly assigned by KYPO.

4.1 KYPO sandbox provisioning

In order to test the two first scenarios, we can use one sandbox definition only. Due to the way that the scripts have been constructed, the router inside the sandbox will have two OpenVPN instances running, one of them acting as a server and another one acting as a client. There is only a slight modification to do to the '*playbook.yml*' file in order to add this feature. As a reminder, we are starting from a basic scenario implementation that is provided by the developers team of KYPO[1].

We will thus create two different connections. The secret keys for both connections must be different otherwise OpenVPN will not be able to launch both instances at the same time. The keys must be placed at the same level as the '*playbook.yml*' file, inside the 'provisioning' folder. They can be generate by launching both of the following commands:

```
$ openvpn --genkey secret client_secret.key
$ openvpn --genkey secret server_secret.key
```

Once the keys have been created, a copy must also be placed inside the virtual machine that will play the role of our outside host.

A first connection, called 'connection1', will act as an OpenVPN server. It will listen on port 4444. The second connection will act as a client, trying to connect to the remote server that will be located on the virtual machine we have setup on our computer. This VM can be seen in Figure 4.1 and has an IP of 10.1.2.59. The VM will have an OpenVPN instance listening on port 1234. In both cases we added extra options that will route any traffic to the network 192.168.2.0/24 through the corresponding VPN tunnel. When doing so, we will however not know in advance which tunnel will be used in order to transfer the

[1] https://gitlab.ics.muni.cz/muni-kypo-crp/prototypes-and-examples/sandbox-definitions/kypo-crp-demo-training

traffic to that network when both connections are active at the same time.

The lines to add to the file are:

```
- name: set up router with OpenVPN
  hosts: router
  become: yes
  roles:
    - name: debian9-update-sources
    - name: openvpn
      openvpn_instances:
      - {
        name: 'connection1',
        server: True,
        tun_ip_local: '172.0.0.1',
        tun_ip_remote: '172.0.0.2',
        secret_file: 'server_secret.key',
        extra_options: ['route 192.168.2.0 255.255.255.0'],
        server_port: 4444,
      }
      - {
        name: 'connection2',
        server: False,
        tun_ip_local: '172.0.0.12',
        tun_ip_remote: '172.0.0.11',
        remote_ip: '10.1.2.59',
        secret_file: 'client_secret.key',
        extra_options: ['route 192.168.2.0 255.255.255.0'],
        server_port: 1234,
      }
```

Listing 3: openvpn role for both client and server creation

At this point, we can upload the project to a public Gitlab repository and add it to the KYPO instance. The exact steps needed to achieve this can be found in the documentation of KYPO. After that we must create a pool, inside of which we will allocate one sandbox. After the allocation is successfully done, we must retrieve two elements:

1. The internal IP address of the MAN node – **192.168.67.70**
2. The "public" IP address of the router inside the sandbox – **100.100.100.3**

4.2 Setup server on VM

Let us first start an OpenVPN server on the virtual machine that is situated outside of KYPO. The configuration used for that is the same as the one that has been shown before. Except that the port will be 1234 and the local and remote tunneling interfaces have IPs of 172.0.0.11 and 172.0.0.12 respectively. Furthermore we add a route to the 192.168.20.0/24 sub-network (which is a network inside the provisioned sandbox) that needs to pass through the tunnel. The resulting configuration looks like this:

```
port 1234
proto udp
dev tun
```

```
ifconfig 172.0.0.11 172.0.0.12
cipher AES-256-CBC
secret client_secret.key

route 192.168.20.0 255.255.255.0

comp-lzo

keepalive 10 60
ping-timer-rem
persist-tun
persist-key
```

We can now test the connection. The entire execution of the different commands can be seen on Figure 4.2.

```
ubuntu@ubuntu:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:cb:1e:0b brd ff:ff:ff:ff:ff:ff
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:d9:5f:96 brd ff:ff:ff:ff:ff:ff
ubuntu@ubuntu:~$ sudo openvpn server.openvpn &
[1] 75003
ubuntu@ubuntu:~$ 2023-05-08 15:14:17 WARNING: Compression for receiving enabled. Compression has been used in the past to break encryption. S
ent packets are not compressed unless "allow-compression yes" is also set.
2023-05-08 15:14:17 Cipher negotiation is disabled since neither P2MP client nor server mode is enabled
2023-05-08 15:14:17 WARNING: file 'client_secret.key' is group or others accessible
2023-05-08 15:14:17 OpenVPN 2.5.5 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Jul 14 2022
2023-05-08 15:14:17 library versions: OpenSSL 3.0.2 15 Mar 2022, LZO 2.10
2023-05-08 15:14:17 TUN/TAP device tun0 opened
2023-05-08 15:14:17 net_iface_mtu_set: mtu 1500 for tun0
2023-05-08 15:14:17 net_iface_up: set tun0 up
2023-05-08 15:14:17 net_addr_ptp_v4_add: 172.0.0.11 peer 172.0.0.12 dev tun0
2023-05-08 15:14:17 Could not determine IPv4/IPv6 protocol. Using AF_INET
2023-05-08 15:14:17 UDPv4 link local (bound): [AF_INET][undef]:1234
2023-05-08 15:14:17 UDPv4 link remote: [AF_UNSPEC]
2023-05-08 15:14:18 Peer Connection Initiated with [AF_INET]10.1.2.192:51457
2023-05-08 15:14:19 WARNING: this configuration may cache passwords in memory -- use the auth-nocache option to prevent this
2023-05-08 15:14:19 Initialization Sequence Completed

ubuntu@ubuntu:~$ ip a | egrep -A5 "tun.\:"
18: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 172.0.0.11 peer 172.0.0.12/32 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 fe80::ba8:cf56:ef23:5749/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
ubuntu@ubuntu:~$ ip route show
default via 192.168.2.1 dev enp7s0 proto dhcp src 192.168.2.246 metric 100
default via 10.1.2.1 dev enp1s0 proto dhcp src 10.1.2.59 metric 100
10.1.2.0/24 dev enp1s0 proto kernel scope link src 10.1.2.59 metric 100
10.1.2.1 dev enp1s0 proto dhcp scope link src 10.1.2.59 metric 100
100.100.100.0/24 via 10.1.2.192 dev enp1s0
172.0.0.12 dev tun0 proto kernel scope link src 172.0.0.11
192.168.2.0/24 dev enp7s0 proto kernel scope link src 192.168.2.246 metric 100
192.168.2.1 dev enp7s0 proto dhcp scope link src 192.168.2.246 metric 100
192.168.20.0/24 via 172.0.0.12 dev tun0
ubuntu@ubuntu:~$ ping 192.168.20.5
PING 192.168.20.5 (192.168.20.5) 56(84) bytes of data.
64 bytes from 192.168.20.5: icmp_seq=1 ttl=63 time=1.88 ms
64 bytes from 192.168.20.5: icmp_seq=2 ttl=63 time=1.59 ms
64 bytes from 192.168.20.5: icmp_seq=3 ttl=63 time=1.54 ms
^C
--- 192.168.20.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.540/1.670/1.880/0.149 ms
```

Figure 4.2: Execution sequence of an OpenVPN server on a virtual machine

The main events have been put inside colored rectangles to make them more easy to see:

1. In the blue rectangle, we first list the interfaces that are active. We can observe that no tunneling interface (interface with a name starting with '*tun*') are present in the output of the command;

2. In the purple rectangle, we start the OpenVPN server and run it in the background;

3. In the red rectangle, we see that the initialization sequence has been completed. This means that the client inside the sandbox has successfully connected to the server and that an encrypted tunnel has been created. If this does not happen, we simply have to restart the service running the OpenVPN client on the router inside the KYPO sandbox;

4. We enumerate all tunneling interfaces by taking all interfaces starting with '*tun*'. We can see that there is indeed an interface that has been created with the name 'tun0'. In the green rectangle, we can confirm that the local address assigned to that interface is '172.0.0.11' which is what was specified inside the configuration file.

5. By looking at the internal routes, we can see in the orange rectangle that a route has indeed been added. This means that every packet with destination 192.168.20.0/24 will pass through the 'tun0' interface;

6. In the cyan rectangle, we try to ping an existing IP address inside that range. Since we get a response we can conclude that the OpenVPN server has correctly been setup.

We do not need to verify on the client side that the connection has been setup. We can, however test if the route has correctly been added to the routing table. We can see from the orange rectangle in Figure 4.3 that a new route has indeed been added. The cyan rectangle in the same figure confirms that we can ping an IP address inside of that network.

```
debian@router:~$ ip route show
default via 100.100.100.114 dev eth1
default via 192.168.20.1 dev eth2
100.100.100.0/24 dev eth1 proto kernel scope link src 100.100.100.3
169.254.169.254 via 192.168.20.2 dev eth2
172.0.0.2 dev tun0 proto kernel scope link src 172.0.0.1
172.0.0.11 dev tun1 proto kernel scope link src 172.0.0.12
192.168.2.0/24 via 172.0.0.11 dev tun1
192.168.20.0/24 dev eth2 proto kernel scope link src 192.168.20.1
192.168.30.0/24 dev eth3 proto kernel scope link src 192.168.30.1
192.168.128.0/17 dev eth0 proto kernel scope link src 192.168.129.106
debian@router:~$ ping 192.168.2.246
PING 192.168.2.246 (192.168.2.246) 56(84) bytes of data.
64 bytes from 192.168.2.246: icmp_seq=1 ttl=64 time=1.18 ms
64 bytes from 192.168.2.246: icmp_seq=2 ttl=64 time=1.10 ms
64 bytes from 192.168.2.246: icmp_seq=3 ttl=64 time=1.67 ms
^C
--- 192.168.2.246 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.103/1.319/1.670/0.250 ms
```

Figure 4.3: Routing table after OpenVPN client instantiated on KYPO

4.3 Setup client on VM

In order to test the case where the client is not on a KYPO instance, we will first deactivate the OpenVPN client on the virtual machine. There are two preliminary steps that must be taken before launching the OpenVPN client on the virtual machine:

1. As has been explained before, we need to add a static route the the OpenStack router. Otherwise, if we try to contact the IP 100.100.100.3 from the virtual machine, the packet will not be able to reach its destination. For the static route, we need to enter the destination address, which is 100.100.100.3, and the next hop, which is 192.168.67.70, the IP address of the MAN node

2. Add a route to the routing table of the virtual machine telling to route all the traffic to 100.100.100.3 to the OpenStack router. This can be achieve by using the following command:

```
$ sudo ip route add 100.100.100.3 via 192.168.67.70 dev enp1s0
```

From this point on, the router inside the KYPO sandbox will be accessible from the virtual machine. These preparation steps will be very similar even if the OpenVPN client (here represented by the virtual machine) is located on a different network across the internet. The information that is shown in the figure is very similar to the case where the VM acted as the VPN server.

```
ubuntu@ubuntu:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:cb:1e:0b brd ff:ff:ff:ff:ff:ff
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:d9:5f:96 brd ff:ff:ff:ff:ff:ff
ubuntu@ubuntu:~$ sudo openvpn client.openvpn &
[1] 76541
ubuntu@ubuntu:~$ 2023-05-08 15:52:04 WARNING: Compression for receiving enabled. Compression has been used in the past to break encryption. S
ent packets are not compressed unless "allow-compression yes" is also set.
2023-05-08 15:52:04 Cipher negotiation is disabled since neither P2MP client nor server mode is enabled
2023-05-08 15:52:04 WARNING: file 'server_secret.key' is group or others accessible
2023-05-08 15:52:04 OpenVPN 2.5.5 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Jul 14 2022
2023-05-08 15:52:04 library versions: OpenSSL 3.0.2 15 Mar 2022, LZO 2.10
2023-05-08 15:52:04 TUN/TAP device tun0 opened
2023-05-08 15:52:04 net_iface_mtu_set: mtu 1500 for tun0
2023-05-08 15:52:04 net_iface_up: set tun0 up
2023-05-08 15:52:04 net_addr_ptp_v4_add: 172.0.0.2 peer 172.0.0.1 dev tun0
2023-05-08 15:52:04 TCP/UDP: Preserving recently used remote address: [AF_INET]100.100.100.3:4444
2023-05-08 15:52:04 UDP link local (bound): [AF_INET][undef]:1194
2023-05-08 15:52:04 UDP link remote: [AF_INET]100.100.100.3:4444
2023-05-08 15:52:06 Peer Connection Initiated with [AF_INET]100.100.100.3:4444
2023-05-08 15:52:07 WARNING: this configuration may cache passwords in memory -- use the auth-nocache option to prevent this
2023-05-08 15:52:07 Initialization Sequence Completed

ubuntu@ubuntu:~$ ip a | egrep -A5 "tun.\:"
19: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 172.0.0.2 peer 172.0.0.1/32 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 fe80::e026:6c0b:8914:9e/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
ubuntu@ubuntu:~$ ip route show
default via 192.168.2.1 dev enp7s0 proto dhcp src 192.168.2.246 metric 100
default via 10.1.2.1 dev enp1s0 proto dhcp src 10.1.2.59 metric 100
10.1.2.0/24 dev enp1s0 proto kernel scope link src 10.1.2.59 metric 100
10.1.2.1 dev enp1s0 proto dhcp scope link src 10.1.2.59 metric 100
100.100.100.0/24 via 10.1.2.192 dev enp1s0
172.0.0.1 dev tun0 proto kernel scope link src 172.0.0.2
192.168.2.0/24 dev enp7s0 proto kernel scope link src 192.168.2.246 metric 100
192.168.2.1 dev enp7s0 proto dhcp scope link src 192.168.2.246 metric 100
192.168.20.0/24 via 172.0.0.1 dev tun0
ubuntu@ubuntu:~$ ping 192.168.20.5
PING 192.168.20.5 (192.168.20.5) 56(84) bytes of data.
64 bytes from 192.168.20.5: icmp_seq=1 ttl=63 time=1.68 ms
64 bytes from 192.168.20.5: icmp_seq=2 ttl=63 time=1.38 ms
64 bytes from 192.168.20.5: icmp_seq=3 ttl=63 time=1.56 ms
^C
--- 192.168.20.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.383/1.542/1.680/0.122 ms
```

Figure 4.4: Execution sequence of an OpenVPN client on a virtual machine

The steps for initiating the client are the same as for the server and can be seen in Figure 4.4. We will not show the adapted configuration file as it is very similar to the one of the VPN server where we adapted the different IP addresses and port number to match what has been specified in the Ansible provisioning script. We can see in the green and orange rectangles, that the IP addresses of the tunneling interfaces have been correctly adapted. The connection has been setup successfully as well.

For the server side on the router inside KYPO, we need to perform an additional step if we want to use the custom route. Indeed, at the moment it is the wrong tunneling interface(tun1) that is used when routing the packets to the 192.168.2.0/24 network. In order to fix this, we will first delete that route and then restart the corresponding connection. After that, we can see that the correct tunneling interface(tun0) is used. The result can be seen in Figure 4.5. The main steps are again showed inside colored rectangles to make it easier to follow the execution steps.

```
debian@router:~$ sudo ip route delete 192.168.2.0/24 via 172.0.0.11 dev tun1
debian@router:~$ sudo systemctl restart openvpn@connection1
debian@router:~$ ip route show
default via 100.100.100.114 dev eth1
default via 192.168.20.1 dev eth2
100.100.100.0/24 dev eth1 proto kernel scope link src 100.100.100.3
169.254.169.254 via 192.168.20.2 dev eth2
172.0.0.2 dev tun0 proto kernel scope link src 172.0.0.1
172.0.0.11 dev tun1 proto kernel scope link src 172.0.0.12
192.168.2.0/24 via 172.0.0.2 dev tun0
192.168.20.0/24 dev eth2 proto kernel scope link src 192.168.20.1
192.168.30.0/24 dev eth3 proto kernel scope link src 192.168.30.1
192.168.128.0/17 dev eth0 proto kernel scope link src 192.168.129.106
debian@router:~$ ping 192.168.2.246
PING 192.168.2.246 (192.168.2.246) 56(84) bytes of data.
64 bytes from 192.168.2.246: icmp_seq=1 ttl=64 time=1.16 ms
64 bytes from 192.168.2.246: icmp_seq=2 ttl=64 time=1.02 ms
64 bytes from 192.168.2.246: icmp_seq=3 ttl=64 time=1.13 ms
^C
--- 192.168.2.246 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.024/1.107/1.161/0.065 ms
debian@router:~$
```

Figure 4.5: Routing table after OpenVPN server instantiated on KYPO

4.4 Configuration generation

Now that we have shown that the first step of the automation works correctly, we will show that the python script automating the configuration generation works as well. In order to test this, we will use the script in order to generate a configuration that is similar to the one shown in Listing 3. We will then compare it to the one we created manually to see if it has worked properly.

Inside this configuration we will assign the name "kypo" to the instance running inside KYPO and "vm" to the one running inside the virtual machine, outside of the KYPO instance. The input configuration that we use is the following:

```
{
    "kypo" : {
        "ip": "100.100.100.103",
        "internal_ip": "192.168.20.0/24",
        "kypo": true
    },
    "vm" : {
        "ip": "10.1.2.59",
        "internal_ip": "192.168.2.0/24",
        "kypo": false
    }
}
```

By setting this file as the input of the python script, we obtain the following results:

1. For "kypo", the configuration file that is generated is the following:

```
[
    {
        "name":"kypo-vm",
        "extra_options":[
            "route 192.168.2.0 255.255.255.0"
        ],
        "server":True,
        "secret_file":"keys/kypo-vm.key",
        "server_port":27604,
        "tun_ip_local":"172.0.0.27",
        "tun_ip_remote":"172.0.0.223"
    }
]
```

   We can observe that it is the same as the one that we specified manually in Listing 3. The only elements that change are the port number and tunnelling ip addresses. This is because of the fact that they are generated randomly inside the script;

2. For "vm", an OpenVPN configuration file has been generated with the following content:

```
remote 100.100.100.103 27604

proto udp
dev tun
ifconfig 172.0.0.223 172.0.0.27
cipher AES-256-CBC
secret keys/vm-kypo.key

route 192.168.20.0 255.255.255.0

comp-lzo

keepalive 10 60
ping-timer-rem
persist-tun
persist-key
```

It can be confirmed that the different values such as the port number and tunnelling IP addresses, correspond to the ones inside the configuration that is used for "kypo". In this case, it is "kypo" which plays the role of the OpenVPN server because it was listed first in our input configuration file.

4.5 Server and client inside KYPO

When both the server and client are inside a KYPO instance, it is only possible to add a VPN tunnel if both sandboxes to connect are inside different KYPO instances. Due to the limited resources at our disposal, we can only start one instance of KYPO at the time and will therefore not be able to test this case in practice. Nevertheless, we have shown that having either a client or a server inside a KYPO instance works well. We could therefore assume, that using the '*openvpn*' role inside multiple separate KYPO instance will work as intended.

As has been introduced, it is not possible to use the created role on two different sandbox definitions inside the same KYPO instance. This is because of the way the networking has been setup. Every router is connected to the MAN node of their pool through a 100.100.100.0/24 network. This means that when we have two pools, we will have two separate networks with the same prefix. When we will try to contact the router of the other sandbox, since the address will have the same prefix as the network it originates from, it will not try to leave it to reach the second network. For example there are two routers in two different sandboxes inside two different pools: router A has IP 100.100.100.103 and router B has IP 100.100.100.3. Router A is connected to its own MAN node and Router B is connected to its own MAN node as well, which is different from the first one. If router A tries to contact 100.100.100.3, the request will never leave the network connecting the router with its MAN node as the address matches the network prefix.

If we still want to create a route between two sandboxes of a same KYPO instance, we need to do that using manual routing techniques.

A first element to take care of is that the interfaces of the MAN nodes that are connected to the 192.168.64.0/18 network are part of an OpenStack security group (*kypo-sandbox-access-sg*) that does not allow them to communicate with each other. In order for them to be able to communicate with each other, we need to add the following three security rules (based on the protocols we want to allow during the communication):

- ALLOW IPv4 tcp from kypo-sandbox-access-sg
- ALLOW IPv4 udp from kypo-sandbox-access-sg
- ALLOW IPv4 icmp from kypo-sandbox-access-sg

When it is done, it is straightforward to be able to route the traffic from one pool to another. It can be done by adding a routing element inside the MAN node that is connected to the sandbox' router. This route must redirect the traffic for a certain target network range to the corresponding MAN node.

4.6 Validation

In this section, we will analyze our results and see to what extend our objectives have been met. A first objective was to implement a federation of cyber ranges. Due to the limited resources, only one instance of the federation could simulate a full KYPO CRP instance, while the other end is a single virtual machine. Using this setup serves as a proof of concept that the federation between different cyber range instances is possible. When implementing this federation mechanism on real cyber ranges, the steps that will be taken are will be the same as the ones that have been explained in the sections above: first use the python script in order to generate the different secret keys and configuration files. Once these elements have been generate, we can either use the configuration inside the sandbox provisioning script (in case of a KYPO CRP instance), or load the OpenVPN configurations directly in the cyber range.

When implementing such federation, it needs to be noted that there is a need to share some information in advance. There are indeed some elements that need to be synchronized between the different scenarios in order to make sure that they will run over the federation. If we are running multiple KYPO instances that are connected using such federation, it is not possible to load one identical scenario on all of the instances. We will need to tailor a new scenario for every cyber range instance. This means that there is some information that needs to be shared across the different scenarios:

- **OpenVPN Secret key** – A first element to share is the OpenVPN secret key that will be used in order to initiate the OpenVPN connection with a peer. It will need to have the same amount of secret keys as the amount of other cyber range instances inside the federation;
- **IP range** – An important element to share between the different instances is the IP range that is used by every cyber range. It is indeed necessary that every instance uses a different IP range in order for every individual network to be reachable over the encrypted tunnels. These IP ranges will also be necessary inside the OpenVPN specification of the gateway. Because it is this file that we will specify that certain IP ranges need to be forwarded over the tunnel, while the rest of them will need to be sent to the internet. This concept is called split tunnelling;
- **Tunnelling interface IP** – This information is needed for the OpenVPN configuration file. As seen before, it is needed to specify the IP address of both the local and the remote tunnelling interface.

These are the two elements that need to be shared across the entire federation. If using the python script that is provided, this shared information will already be present inside the different folders corresponding to each cyber range. For the rest, every scenario can be implemented independently from each other. Depending on the use case of such federation, it will also be useful to share the IP addresses that have been assigned to the different elements inside the network. In some cases, such as a "Capture The Flag" type implementation, we would not like to share the entire network topology in advance in order for the participant to be in a situation that is as close to the reality as possible. In other cases however, such as in some penetration testing instances, we will want to know the topology of the other instances in advance.

A second objective of this work, was to make the communication inside the federation as secure as possible. The main component inside the federation that will define the security is the security of the OpenVPN tunnel. As explained before, OpenVPN adds encryption at the Application layer by using SSL/TLS. It is important to consider the security of this encrypted tunnel as well, since it will be routed over the internet that is considered to be insecure. If a communication link is intercepted, or the VPN tunnel is corrupted, it could lead to leaks of sensitive information, depending on the use case of the cyber range federation. OpenVPN is however considered to be secure as it adds authentication on top of the encryption layer, as long as the secret keys are exchanged in a secure manner before the setup of the VPN tunnel.

A final objective for this implementation was to automate the process as much as possible. We have seen from the previous sections, that we can simply use the created role inside the provisioning script in order to launch an OpenVPN instance (either a server or a client) on a router. For this to work, we can generate the different secret keys manually as well as define the IP addresses that we want to use for the tunnelling interfaces. Another solution, is to use the Python script that has been created in order to generate all of these automatically. This will be helpful especially if there are more than 2 cyber ranges inside the federation, or if the ranges do not all use the KYPO cyber range platform.

Chapter 5
Conclusions

The objective of this master thesis was to implement a way of interconnecting multiple cyber range instances in a way that is as automated as possible. The need for such federation mechanism is because every cyber range can have its own specialization and location. For this reason, we want to be able to communicate from one range to another in a way that is as transparent as possible for the user.

We have started this work by presenting the characteristics of a cyber range and the different technologies that would be used in order to build the federation. These technologies mainly included VPN solutions. From that analysis, we have concluded that OpenVPN would be the best solution. As for the cyber range platform, KYPO CRP has been chosen mainly due to the fact that it is open-source and actively maintained.

Once the technology has been used, we have proceeded in two steps in order to design the automation process: first analyze how we can create such VPN tunnel manually and then try to automate that process using KYPO's provisionning language. For the first step, we have shown that creating an OpenVPN tunnel can be done manually in a few steps, with the hypothesis that both instances that would like to communicate in a secure manner have a shared secret key.

The last step that had to be performed was to translate these steps into an Ansible playbook, which will then be used by KYPO in order to provision the sandbox. We have furthermore gone a step further and automate, by using a Python script, the creation of the variables that the playbook needs in order to provision the sandbox.

We can conclude this work by stating that the goal of this master thesis, which has been presented at the beginning, has been fulfilled. We have shown that the sandbox that is created with our automation script, is able to successfully launch both a VPN client and VPN server. Once that the sandbox has been created, there are no further steps that must be taken. This means that we have provided a way to fully automate that process. However, the simulation that has been made is all on a single machine. We would therefore still need to perform the same steps while using multiple instances of KYPO CRP, that are located at different locations. As has been explained during this work, this should not be a problem, as long as we are able to create a route between the different VPN gateways.

Bibliography

[1] Barker, E., Dang, Q., Frankel, S., Scarfone, K., Wouters, P.: Guide to IPsec VPNs. Tech. rep. (Jun 2020)

[2] CISCO: Global cloud index projects strong multicloud traffic growth (Feb 2018), https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2018/m02/global-cloud-index-projects-cloud-traffic-to-represent-95-percent-of-total-data-center.html, accessed: 2023-01-04

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

[4] Endsley, M.: Toward a theory of situation awareness in dynamic systems. Human Factors: The Journal of the Human Factors and Ergonomics Society 37, 32–64 (03 1995)

[5] Graziano, A., Borines, S.R.: User emulation cyber range (Oct 2022), https://cyberstartupobservatory.com/user-emulation-cyber-range/, accessed: 2023-01-05

[6] Hannay, J.E., Stolpe, A., Yamin, M.M.: Toward AI-Based Scenario Management for Cyber Range Training. Springer (2021)

[7] Maxstead, M.: VPN Types and the OSI Model - VPNReviewz.com — vpnreviewz.com. https://vpnreviewz.com/vpn-types-and-the-osi-model/, [Accessed 07-May-2023]

[8] Messina, G.: Types of cyber ranges compared: Simulations, overlays, emulations and hybrids (Apr 2021), https://resources.infosecinstitute.com/topic/types-of-cyber-ranges-compared-simulations-overlays-emulations-and-hybrids/, accessed: 2023-01-05

[9] Mica R., E., Michelle M., R.: Training for situation awareness in individuals and teams, chap. 16, pp. 349–366. CRC Press (2000)

[10] Nakata, R., Otsuka, A.: Cyexec*: A high-performance container-based cyber range with scenario randomization. IEEE Access 9, 109095–109114 (2021)

[11] National Initiative for Cybersecurity Education Cyber Range Project Team: The cyber range: A guide. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD (2020)

[12] Nock, O., Starkey, J., Angelopoulos, C.M.: Addressing the security gap in iot: Towards an iot cyber range. Sensors (Basel, Switzerland) 20(18), 1–19 (2020)

[13] Oikonomou, N., Mengidis, N., Spanopoulos-Karalexidis, M., Voulgaridis, A., Merialdo, M., Raisr, I., Hanson, K., de La Vallee, P., Tsikrika, T., Vrochidis, S., Votis, K.: Echo federated cyber range: Towards next-generation scalable cyber ranges. In: 2021

IEEE International Conference on Cyber Security and Resilience (CSR). pp. 403–408 (2021)

[14] Peratikou, A., Louca, C., Shiaeles, S., Stavrou, S.: On federated cyber range network interconnection. In: Ghita, B., Shiaeles, S. (eds.) Selected Papers from the 12th International Networking Conference. pp. 117–128. Lecture Notes in Networks and Systems, Springer (Jan 2021), 12th International Network Conference, INC 2020 ; Conference date: 21-09-2020 Through 21-09-2020

[15] Priyadarshini, I.: Features and architecture of the modern cyber range: A qualitative analysis and survey (2018)

[16] Ross, R., Pillitteri, V., Graubart, R., Bodeau, D., McQuaid, R.: Developing cyber-resilient systems: A systems security engineering approach. Tech. rep., Gaithersburg, MD (2021)

[17] Sava, J.A.: Global vpn market size 2019-2027 (Jan 2023), `https://www.statista.com/statistics/542817/worldwide-virtual-private-network-market/`, accessed: 2023-01-04

[18] Statista: Internet and social media users in the world 2022 (Sep 2022), `https://www.statista.com/statistics/617136/digital-population-worldwide/`, accessed: 2023-01-04

[19] Ukwandu, E., Farah, M.A.B., Hindy, H., Brosset, D., Kavallieros, D., Atkinson, R., Tachtatzis, C., Bures, M., Andonovic, I., Bellekens, X.: A review of cyber-ranges and test-beds: Current and future trends. Sensors 20(24), 7148 (Dec 2020)

[20] Valamis: What is a learning management system: Lms overview (Sep 2022), `https://www.valamis.com/hub/what-is-an-lms`, accessed: 2023-02-02

[21] Vykopal, J., Ošlejšek, R., Čeleda, P., Vizváry, M., Tovarňák, D.: Kypo cyber range: Design and use cases. pp. 310–321 (2017)

[22] Vykopal, J., Vizvary, M., Oslejsek, R., Celeda, P., Tovarnak, D.: Lessons learned from complex hands-on defence exercises in a cyber range. In: 2017 IEEE Frontiers in Education Conference (FIE). pp. 1–8 (2017)

[23] Yamin, M.M., Katt, B., Gkioulos, V.: Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. Computers & Security 88, 101636 (2020)