Cyber range external monitoring solution

Enabling external users of a cyber range with monitoring capabilities using SSH tunnels and Zabbix.

Allan Muranovic
Dr. Jérôme Dossogne PhD

Research and Development project owner:
Dr. Jérôme Dossogne PhD

Master thesis submitted under the supervision of
Dr. Jérôme Dossogne PhD

in order to be awarded the Degree of
Master in Cybersecurity
System Design

Academic year
2021 – 2022

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.

30/05/2022

Title: Cyber range external health monitoring solution

Author: Allan Muranovic
Dr. Jérôme Dossogne PhD
Master in Cybersecurity – System Design
Academic year: 2021 – 2022

Abstract

The popularity of the cyber range has been increasing over the last few years and is not likely to stop growing. In some cases, companies are using them to organise cyber exercises for any participant who wants to join in. In such situations, it is common for the company to use an external instructor to guide the participants. These instructors and participants are considered external to the infrastructure, and generally do not have many rights over it (compared to an administrator who can modify, add or delete a machine). In addition to this, they often need to set up a VPN or go through a "pivot" machine in order to access the infrastructure, which makes the machines not directly accessible to this category of person.

These two elements have as a consequence that it is not easy for these people to detect when the infrastructure has a problem (a machine is no more reachable for example).

In this paper, in order to solve this problem, we propose a monitoring solution for users in such situations. The proposed solution combines the Zabbix monitoring tool, and its agentless features, with SSH tunnels to reach machines that are not directly accessible.

Our solution allows, not only to know directly when a machine is no longer available via an alert system, but also to monitor the elements of a machine to ensure that it is still in its initial state (as at the beginning of the cyber exercise).

Keywords: Cyber range, monitoring, external, SSH tunnels, Zabbix, pivoting, infrastructure, health

Acknowledgements

First of all, I would to thank the supervisor of this study: Pr. Jérôme Dossogne. I thank him for the time he spent guiding me, clarifying my ideas, making this LateX template and for answering all my questions that might have seemed legitimate in my head but a but foolish when said out loud...

He helped me through the whole process from the research of my subject till the very last sentence, with the greatest possible pedagogy.

I would also like to thank my family, who supported me and kept giving me all the time and strength I needed to carry out this research.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
DMZ	Demilitarized Zone
IaC	Infrastructure as code
IOC	Indicator of compromise
K.P.I.	Key performance indicator
VM	Virtual Machine
VPN	Virtual Private Network

Chapter 1
Introduction

1.1 Motivations & Context

Today, we are in a world where digitalization is accelerating every day. As pointed out in a study by the European Union [11] on the services offered by private companies, they are increasingly conducting business electronically. The majority of them offers their products on the internet, expending the use of cloud computing and even social networks.

Unfortunately, as more and more services are accessible via the internet, the number of security breaches in these services has also increased. As reported in a study by Forbes [12], the number of data leaks and breaches has hiked in the past few years. Forbes also estimates that these numbers will continue to rise in the coming years.

With the increase in the number of cyber attacks, it is necessary to train cyber specialists in detecting them. One solution that is becoming progressively popular these last years, as shown in the literature study "cyber ranges and security testbeds: Scenarios, functions, tools and architecture (2020-01)" [24], is the usage of cyber ranges. These cyber ranges can be used, among other things, to host exercises allowing these specialists to improve their skills.

In our personal experience, we have found that some entities may organise a cyber exercise using external instructors to lead the participants. These instructors have, in general, no particular rights on the infrastructure (not like an administrator or creator) but can possess the solution of the exercise as well as the necessary credentials.

These instructors guide participants who, usually, have as few rights as the instructor without any knowledge or information about the exercise.

In some cases, when a machine is only accessible by pivoting from another, or when a VPN configuration is required to access the infrastructure, monitoring the infrastructure becomes more complicated for a external person to the infrastructure such as these participants or instructors.

The purpose of this monitoring is not to receive the same data as the administrator, but simply to allow the instructor or participant to know that a machine is still accessible and/or that it is still in its initial state (knowing that it is in the initial state ensures that the machine is still usable for the exercise).

Finally, as it is complicated to monitor these machines, some tricks, such as pinging the machine continuously, are used. But these tricks can be quite inefficient. And on top of that, they don't allow us to know if a user has modified a certain machine.

By having this information, participants and instructors could know more efficiently when and if there is a problem with a machine and therefor contact the administrator who could then target the problem directly.

And, when a problem occurs, if it is rapidly determined, it allows the administrators to have the information directly, and avoid wasting the participants and instructors time. Having such a solution would potentially save time and therefore improve the efficiency of

such an exercise for all the parties involved; the owner of the infrastructure as long as the external users.

1.1.1 Problem statement

In this paper, we are focusing on finding a solution allowing third-party of an infrastructure (as a participant and/or instructor of a cyber exercise hosted through a cyber range) to monitor the needed elements of every wanted devices from the infrastructure. The monitoring solution should alarm the user as soon as the state of a device changes.

1.2 Project statement & contributions

For this research, in order to find a solution for the problem stated in the previous section 1.1.1, we made a state-of-the-art of current monitoring techniques of cyber ranges, popular monitoring solutions and health probes techniques in the cloud computing. We did not find any solution focusing on a third-party (such as an instructor and/or participant) point of view allowing to monitor the wanted elements.

Indeed, the found solution did not allow someone, without administrator right (such as the right to install an agent on a device for example) on the infrastructure, to monitor the elements they want.

Additionally, we determined the interesting elements in a device for a participant and/or instructor of a cyber exercise.

We did so by, first, looking at popular pentesting framework (which are often used as a guide for such an exercise), and by collecting the elements indicated as "to look at" by the latter.

Second, we looked at what the administrator of an infrastructure uses in order to know if its state has been modified or not.

These two points of view allowed us to gather the elements to monitor in priority for an instructor/participant in a cyber exercise.

After noticing that there was no offered solution for our problem, we looked at free-to-use monitoring solutions.

We browsed through a number of more or less complete monitoring tools, and settled on the three that seemed the most complete/relevant and popular: ELK Stack, Prometheus and Zabbix.

After that, by comparing their list of features, we chose Zabbix which seemed to tick the most boxes.

Subsequently, we used the agentless monitoring feature of Zabbix and combined it with SSH tunnels that allows us to monitor machines that are not directly accessible (behind a pivot or a VPN for example.)

Finally, we automated the deployment of Zabbix using IaC with Terraform and ansible. Once the server was configured, we used python3 to create the SSH tunnels and also to create the monitoring tasks on the Zabbix server, deployed earlier, through its API.

The code needed to deploy our solution is available in annex B.

1.3 Organization of this document

After this chapter, the document is structured as follows:

Chapter 2: Definitions, Literature review and state of the art (SotA), in which we define the concepts and technologies we used, as well as the cyber range, cyber range monitoring, monitoring solution, health probes SotA.

Chatper 3: Project's mission, objectives and requirements, in which we define our mission, our objectives, the requirements our solution needs to fill, the requirements covered or not by the SotA and the project scope as well.

Chapter 4: Experimentation & Implementation, in which we explain the experiments we made that lead us to the implementation of our solution.

The experimentation is followed by **Chapter 5: Experiment's output Analysis**, in which we analyse our solution and if/how it fills our needs.

We discuss the cybersecurrity analysis of our project in **Chapter 6: CyberSecurity analysis of the proposed solution** as well.

After analysing our solution, we compared it with the SotA, **in Chapter 7: Discussion**, in order to explain the limitation of validity and describe the future work that could be done to improve it.

Finally, we shared our conclusions in **Chapter 8: Conclusions**.

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

This chapter aims to define each key concept and terms that are later used in this paper. Following the definitions, we describe the SotA of the cyber range and its monitoring solutions.

After reviewing the academics papers, we will describe popular monitoring solutions and the technologies used for this research.

Finally, we've studied what an attacker is looking for in an infrastructure during pen test and what a defender uses as indicator of compromise in the same infrastructure. We then used the view from both sides to define the elements to be monitored.

2.1 Definitions

2.1.1 Cyber range

We define a cyber range following the NIST [27] agency definition:

definition

"Cyber ranges are interactive, simulated representations of an organization's local network, system, tools, and applications that are connected to a simulated Internet level environment. They provide a safe, legal environment to gain hands-on cyber skills and a secure environment for product development and security posture testing".

2.1.2 Cyber exercise

We define a cyber exercise as any kind of exercise hosted in a cyber range.

As explained in section 2.1.1, such an exercise may have different purpose.

2.1.3 Penetration testing (Pen test)

When reviewing an infrastructure by simulating an attack on it, a cyber specialist often follows a "Penetration testing" framework. We follow the NIST definition as well: [32]

definition

"Security testing in which evaluators mimic real-world attacks in an attempt to identify ways to circumvent the security features of an application, system, or network. Penetration testing often involves issuing real attacks on real systems and data, using the same tools and techniques used by actual attackers. Most penetration tests involve looking for combinations of vulnerabilities on a single system or multiple sys-

tems that can be used to gain more access than could be achieved through a single vulnerability."

2.1.4 Monitoring software

Before reviewing the monitoring solutions for a cyber range, we first need to define it.

Following Techopedia [39] proposition of a monitoring software :

definition

"Monitoring software observes and tracks the operations and activities of users, applications and network services on a computer or enterprise systems. This type of software provides a way to supervise the overall processes that are performed on a computing system, and provides reporting services to the system or network administrator. "

2.1.5 Metrics

According to the popular monitoring software [34] Prometheus:

definition

"In layperson terms, metrics are numeric measurements, time series mean that changes are recorded over time. What users want to measure differs from application to application. For a web server it might be request times, for a database it might be number of active connections or number of active queries etc."

2.1.6 Exporter

We consider an exporter as any tool that aims to export wanted metrics from a given device to another one. An exporter can be agent based (i.e. an agent to install on the machine on which we retrieve the data) or agent less (nothing to install on the target machine).

2.1.7 Virtual Private Network

In some cases, cyber ranges are not directly available on the Internet, a VPN configuration might be required.

We follow the wikipedia [41] proposed definition :

definition

"A virtual private network (VPN) extends a private network across a public network and enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network. The benefits of a VPN include increases in functionality, security, and management of the private network. It provides access to resources that are inaccessible on the public network[...]"

2.1.8 Pivot

In some infrastructures, with or without VPN, it is necessary to access an intermediate machine before being able to reach another hidden behind it.

In fact, this happens a lot when the one accessing the infrastructure is not one of the sysadmin/owner of this same infrastructure.

However, even if the one accessing the infrastructure is a sysadmin, he may set up "jumpboxes", "control towers",..., a unique machine through which he needs to go through in order to administrate any other machine of the infrastructure.

These set up devices, allow, for instance, to restrict access to the administration services of a server only from a single device by banning every IP addresses to access its port 22[1] except the one of the set up device.

For example, as illustrated in figure 2.1, when a client wants to access the Server A, B or C, he has to go through the publicly accessible server. We call it the "pivot" (which may sometime be the "jumpbox", "control tower",...). Multiple pivots may be needed to access a given device as well.

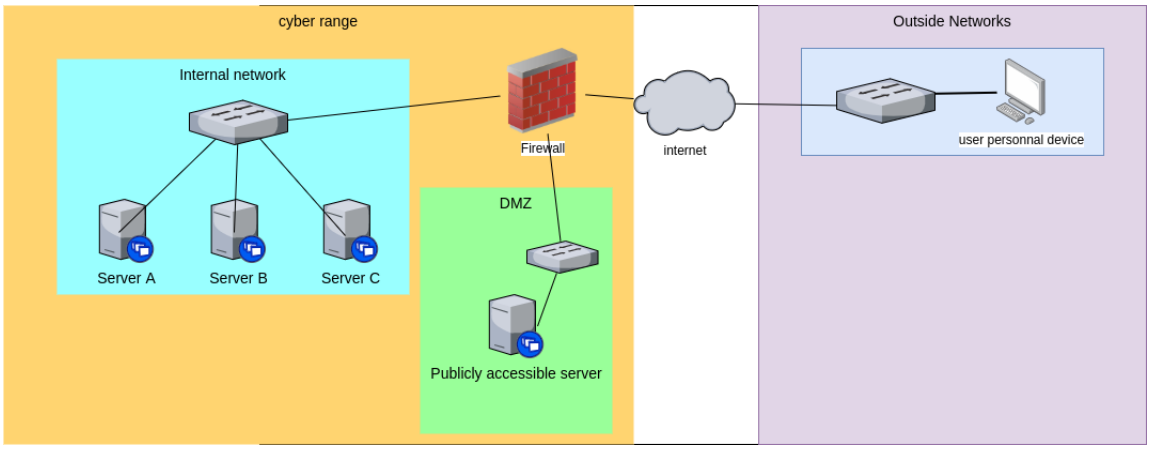

Figure 2.1: Simple illustration of a pivot

2.1.9 Demilitarized zone (DMZ)

As we saw in figure 2.1 the publicly accessible server is placed in a DMZ.

As NIST agency defines [31] it :

definition

"Perimeter network segment that is logically between internal and external networks. Its purpose is to enforce the internal network's information assurance policy for external information exchange and to provide external, untrusted sources with restricted access to releasable information while shielding their internal networks from external attacks."

[1] The commonly used port for SSH accesses.

2.1.10 SSH tunnel

To operate network services over an unsecured network, the **S**ecure **SH**ell protocol is well known and often available out of the box on popular UNIX distribution such as Ubuntu, Debian, CentOS,...

And, in some cases, it is possible to transmit data using an SSH tunnel.

As defined by the SSH protocol maintainer and developer, SSH security [38] :

definition

"SSH tunneling is a method of transporting arbitrary networking data over an encrypted SSH connection. It can be used to add encryption to legacy applications. It can also be used to implement VPNs (Virtual Private Networks) and access intranet services across firewalls."

With the figure 2.2, *SSH security* propose a simplified overview of a secure connection over the untrusted network, which is established between an SSH client and an SSH server. [38]

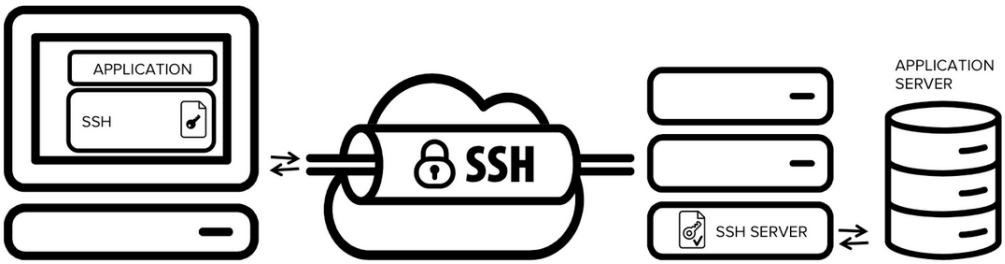

Figure 2.2: Simplified picture of ssh tunneling from ssh.com

2.1.11 Summary of the relationships between the defined concepts

In order to study a cyber range monitoring solution for an outsider such as an event instructor or participant, we first needed to define what a cyber range and a cyber exercise are. Additionally, when a cyber exercise is offered by an entity as a public challenge, it often requires to get accessed through a special VPN or a given pivot for security reasons.

After that, we defined our conception of a monitoring software and its components which we will later combine with SSH tunnels in order to monitor a cyber range.

The figure brings together all the elements defined above.

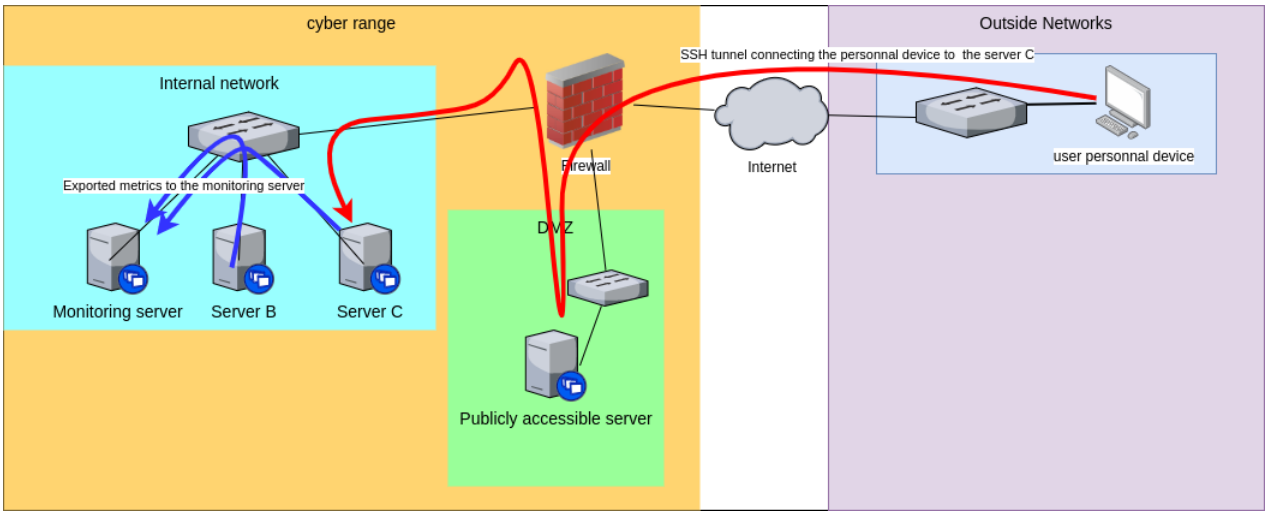

Figure 2.3: Figure containing all the defined concepts.

2.2 State of the art and related works

2.2.1 Cyber range

We defined what a cyber range is in section 2.1.1.

Additionally, we read a literature review made by Muhammad Mudassar Yamin, Basel Katt and Vasileios Gkioulos [24]. These researchers reviewed 100 papers from 2000 to 2020. The papers are all about elements of a cyber range such as its monitoring, creation, maintaining,...

Their study showed that the number of researches taking interest in cyber ranges, as well as their usage, keep growing each year. One of their conclusion is that cyber ranges are gradually becoming popular tool to train cyber specialist.

Also, as defined by Shabeer A., Nicolò M. and Prinetto P. [36], there are 3 main different approaches to create a cyber range topology: a physical cyber range, a virtual cyber range or a hybrid one (which has virtual components and physical one as well).

These different types of cyber ranges can be deployed using diverse tools such as Proxmox, VMware, Emulab... [24].

In our case, we focused on the external person point of view, which has no information about the infrastructure type. So, we want a monitoring solution that does not require to know the kind of cyber ranges it works on.

Finally, as the NIST agency explains it, a cyber range can be made for:

(1) Professionals "from diverse groups such as information technology, cyber-security, law enforcement, incident handlers, continuity of operations, and others to use cyber ranges to improve individual and team knowledge and capabilities" [30].

(2) Students who "can use cyber ranges to apply knowledge in a simulated network environment, develop cyber skills, work as teams to solve cyber problems, and prepare for cyber credentialing examinations" [30].

(3) Educators who "can use cyber ranges as a classroom aide or instruct or assess students virtually" [30].

(4) Organizations which "can use cyber ranges to evaluate their cyber capability, test new procedures, train their team on new organizational and technical environments and protocols before they are introduced into the organizational environment and expand personnel abilities" [30].

2.2.2 Cyber range monitoring

In our literature review regarding the monitoring of a cyber range, we found studies focusing on every aspect of it, some were focusing on data collection through different analysis modules [3, 7, 14], on the dashboard development within the infrastructure [2, 9, 14, 21], some were focusing on the resources monitoring like the CPU or the memory [23], or even by using layer protocols in order to collect the data [2, 22, 40, 42].

The majority of these monitoring solution uses the same tools (each one has a different purpose) such as Wireshark [37][2], Sigar API [9][3] or Nagios [7][4].

[2] https://www.wireshark.org/
[3] https://github.com/cb372/metrics-sigar
[4] https://www.nagios.com/

All these solutions were relatively out of scoop for our problem as they aim to monitor the infrastructure **(1)** for the sysadmins **(2)** within the infrastructure (a direct access to the devices are required) or even **(3)** focusing on a given task (unrelated to ours) such as monitoring the resource usage of an infrastructure.

In addition to the first founded researches, we found some closer to our case but still far from our objective. They focus on either the participants, elements we are also looking at monitoring, or agentless monitoring.

In the "Design of Cyber Warfare Testbed" [8] paper, the authors Chandra and Mishra propose a realistic cyber warfare testbed using a virtual infrastructure, commodity servers and open-source tools.

Regarding the monitoring of the cyber exercise, they collect the data after the exercise has been completed. They do not offer a solution to monitor the system in real time for a sysadmin nor a third-party[5].

In a second paper, "Scalable Agentless Cloud Network Monitoring" [5], the authors focus on the implementation of a solution allowing an agentless monitoring of an infrastructure using Simple Network Management Protocol (SNMP) which is then read through Prometheus services using Graphana as a Dashboard. Sadly, this solution allows the monitoring on an infrastructure from within only and does not allow to retrieve the information we want.

In a third paper, "Using an emulation testbed for operational cyber security exercises" [1], the researchers set up[6] a monitoring system in their cyber range by configuring a Zabbix server within the infrastructure and by installing an agent on every devices as well.

In this research, they found Zabbix to be well suited for the job. It allowed them to retrieve the wanted data, visualize it and alert the sysadmin when needed.

However, Zabbix required two things: **(1)** to have a direct access to all the monitored devices and to **(2)** install an agent on every monitored devices (thus modifying their state).

Finally, the solution was made for the administrator not for someone like a participant.

In another research, "Towards the Monitoring and Evaluation of Trainees' Activities in Cyber Ranges" [4] the authors focus on monitoring the participants actions to give them a score. This is the only research we have found focusing on the participants. But the goal was then to monitor their acts, not to allow them to actually monitor anything.

In a fifth one,"Monitoring of Cyber Security Exercise Environments in Cyber Ranges" [23], the researcher offered a monitoring solution for the Swedish army cyber range (CRATE).

During this research, the author created a survey and questioned different actor of a cyber exercise (such as the creators, the participants and the maintainers) to identify the key points that needed to be monitored for each of them. However, the author suggested a built-from-scratch solution from within the range (which requires a direct access). She then determined the elements the participants wanted to monitor by conducting a survey with only 12 people, and did not create a monitoring solution that could be used by the participants as well.

[5] As a participant for example, which has not the credentials to directly access and manage the infrastructure.

[6] And automated its deployment

Lastly, among the elements to be monitored that we identify later in section 2.3.8.1.4, some are similar to those defined by her survey (such as the firewall stats and running processes).

As a final step, we also looked at the SotA of health check techniques used in the cloud that could be applied here.

For example, in "Technical Health Check For Cloud Service Providers" [6], the authors create Key Performance Indicator (KPI) from raw operational data retrieved in the infrastructure. This allows them to look at the failure rate, utilisation rate, etc. of their infrastructure. However, the solution needs to have special rights in order to be able to retrieve this information.

Unfortunately, as in the cyber range monitoring, we did not find any health check techniques/methodologies matching our criteria (checking the wanted criteria of a machine that does not belong to us).

2.2.3 Conclusion

There are advances in the monitoring system for cyber ranges. A lot of potential needs of the infrastructure owner have been covered, but they are all done from the cyber range creator or system administrator with direct accesses[7] point of view.

However, we found a paper [23] which addresses the needs from the participant's perspective. The author even interviewed 12 participants to find out about their needs. And among our elements defined later in section 2.3.8.1.4, there are several elements that we determined which were determined by his survey as well.

We have not found any paper focusing on **(1)** offering a monitoring solution for a participant who **(2)** has the particularity of not having a direct access to the infrastructure (as a sysadmin has). The complete details about our needs are defined in section 3.

2.2.4 Review methodology

During our state of the art review, we went through articles about cyber ranges, security testbed[8], cyber ranges monitoring, monitoring, agentless monitoring, health check, health probes (which are methods used in the cloud to check the "health" of a device) and non-directly accessible infrastructure monitoring available on Dblp, Mendeley, Google Scholar, Researchgate and other popular scientific paper platforms. When we found an interesting article, as the literature review [24] for example, we went through its bibliography to find other relevant papers. After that, we looked for other articles that were not in the literature review, and for articles that were published after the literature review.

Finally, we looked at the exact purpose of each cyber ranges technologies, popular monitoring tool and the use cases before having a clear definition of our problem and its goal.

[7] Here we consider administrator who can modify the infrastructure as he wants, not someone who has the administrator credential of a given machine within the infrastructure.

[8] Which we found out to be another term for cyber range.

2.3 Implementations, standards, protocols and technologies

In this section, we develop the technologies studied or used during this research.

2.3.1 Popular monitoring tools

As founded in section 2.2.2, there already are researches aiming to develop a monitoring solution for a cyber range with each following a given goal. None of these solution can be applied to our case.

This made us explore available products that could be suited to our needs. During this research, we looked at publicly available monitoring tools and solution found in researches papers as well (such as the one found during the literature review [24] focusing on the monitoring point of a cyber range).

At the end, we found no, free-to-use, monitoring solution meeting our needs defined later in section 3.

But we found popular tools that could be adapted to our needs: Prometheus [34], ELK stack [43] and Zabbix [45].

Finally, the popular copyrighted and not free-to-use monitoring solution (such as SolarWinds, Auvik, Datadog,... for example) are not covered here. We considered paying their services out of scoop for this academic research.

2.3.2 Prometheus

As defined on their official website [34] page :

"Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company.[...]"

In resume, Prometheus is a raw metrics retrieving tool, with a lot of available exporters. Once the raw metrics are retrieved, a tool like Graphana is used to parse and visualize the data.

Finally, HG [15] insights[9] found 13 000 companies, among which we can find General Motors and AT&T, that are using Prometheus as a monitoring software.

2.3.2.1 Components

The Prometheus components are described on their website [33].

The ecosystem is composed of multiple components, which are mostly written in Go, many of which are optional:

- the main Prometheus server which scrapes and stores time series.
- client libraries for instrumenting data application code.
- a push gateway for supporting short-lived jobs.
- special-purpose exporters for several services.
- various support tools.

[9] HG insights is a company which retrieves information on the software used by companies. It is the only source of information we found estimating the number of users for a given software. These number have to be taken carefully.

2.3.2.2 Architecture

In figure 2.4, we can see an overview of the Prometheus architecture.

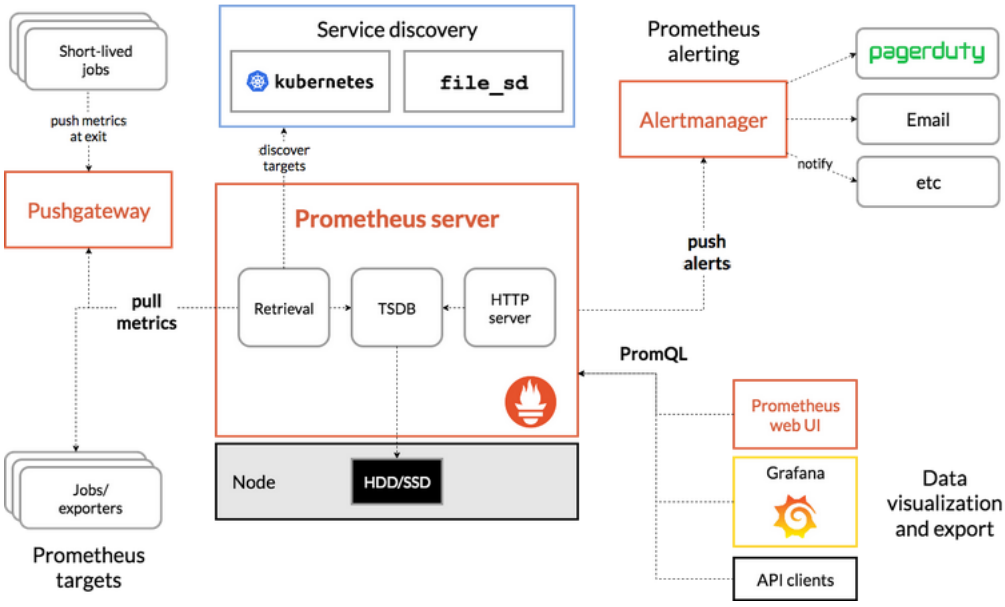

Figure 2.4: Prometheus architecture and its main components. [33]

As it is a free software, we tested the Prometheus server, alert manager and its data visualization. Regarding the exporters, as any contributor can implement its own, there are many of them. Each of the proposed exporter aims to fill a given task such as monitoring an Apache server for example.

2.3.3 ELK stack

As explained on their website [10]:

" "ELK" is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana lets users visualize data with charts and graphs in Elasticsearch. "

And, still according to HG insights, there are around 21 000 companies (which includes Boeing and Verizon for example) using the ELK stack as monitoring solution for their infrastructure.

2.3.3.1 Components

To put it in simple terms, the Elasticsearch and Logstash element act as Prometheus. The Kibana tool is comparable to Graphana in a way that it allows to parse the data and generates useful easy-to-read graphs.

Later, the ELK stack solution was combined with Beats Stash, which is an agent to install on an target host in order to export the needed data to the wanted server. Beats acts like the data exporter of the monitoring solution.

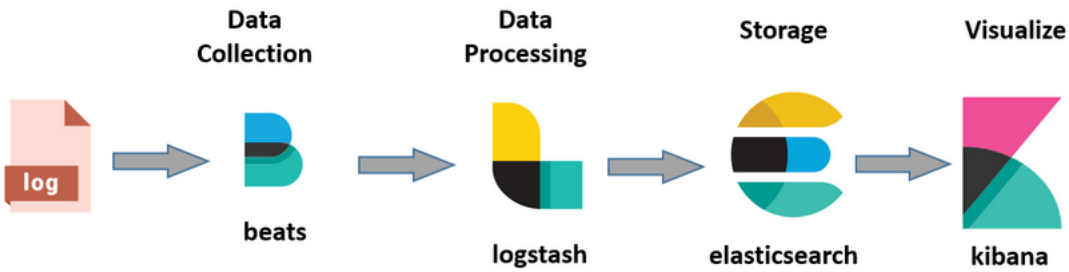

Figure 2.5: Illustration of ELK stack main components.

Finally, ELK stack offers an open-source free version and a paid one as well allowing the user to obtain more services such as reporting, alerting, detection alert external notifications and actions and many other elements depending on the selected package[10]. Even if they are not as complete, these elements are freely provided in Prometheus.

2.3.4 Zabbix

As described in their official documentation [44]:

"Zabbix is an enterprise-class open source distributed monitoring solution.

Zabbix is a software that monitors numerous parameters of a network and the health and integrity of servers, virtual machines, applications, services, databases, websites, the cloud and more. Zabbix uses a flexible notification mechanism that allows users to configure e-mail based alerts for virtually any event. This allows a fast reaction to server problems. Zabbix offers excellent reporting and data visualization features based on the stored data.

[...]All Zabbix reports and statistics, as well as configuration parameters, are accessed through a web-based frontend. [...]

Zabbix is free of cost. Zabbix is written In C and PHP and distributed under the GPL General Public License version 2. "

Zabbix is an "all-in-one" solution. It means that Zabbix includes a data visualization dashboard (as Graphana or Kibana), the configuration of exporter and includes a guest dashboard as well[11].

In addition, Zabbix provides a guest interface. Which means that a guest would be allowed to visualize some data without being able to modify or access a sensitive information as a target password. And, as ELK stack, Zabbix offers an API allowing the automation of the monitoring configuration.

Finally, according to HG insights, Zabbix is currently used by 30 000 companies (which includes Apple and Zoom).

2.3.5 Prometheus vs ELK stack vs Zabbix

In this section, we compare each solutions reviewed features.

Overall, the set of features offered by each solution differs very little. And if we believe HG insights numbers, Zabbix is the most used software for companies.[12] And during our

[10]ELK stack offers a Standard, Gold, Platinum and Entreprise package with different prices.

[11]The dashboard allows an outsider to visualize the data the admin configured, but not to do anything else without explicit rights.

[12]In term of companies using it, not the number of people.

literature review, we have found a recent paper [1] using Zabbix in a cyber range, one using Prometheus on a cloud infrastructure while no similar project was found with ELK.

The features offered by these solutions are the same, the form differs a little and that is what led to favour the use of Zabbix.

For example, even if Prometheus does have a community made agent less exporter, we have not found any that has been maintained in the past 5 years. The founded exporter are limited as well, we can't export the list of users, meta-data of a file or other pertinent elements that we need.

And finally, none of these solutions allows an out of the box agentless monitoring of an non directly accessible infrastructure.

Thus, the next table is a comparison of these features.

Comparaison elements	**Prometheus**	**ELK stack**	**Zabbix**
Alerting notifications systems	Yes	Yes	Yes
Open source	Yes	Yes	Yes
Free to use	Yes	Yes	Yes
Agentless monitoring	No	No	Yes
Guest Dashboard	No	No	Yes
Monitoring behind a pivot with installable agent	Yes	Yes	Yes
Agentless monitoring behind a pivot	No	No	No

2.3.6 Terraform

In this academic research, in order to create a virtual machine, we used Terraform.

As explained on their official website [13], Terraform is "an open-source infrastructure as code software tool that provides a consistent CLI workflow to manage hundreds of cloud services. [...]"

When combined to Microsoft azure[13], Terraform allows an easy way to create/manage/destroy a virtual machine.

2.3.7 Ansible

After creating a VM using Terraform, we configured it using the popular provisioning tool.

Ansible is a tool created and maintained by the RedHat company that describes it as "a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs." [35].

2.3.8 Elements to monitor

In order to find out which elements are necessary for the proper functioning of an exercise in a cyber range, we looked at the point of view of the defenders as well as the attackers.

[13]Which offers a free trial for a ULB student account.

2.3.8.1 Attackers side

To determine what an attacker is looking for, thus determining the elements that need to be monitored, we went through 3 popular pen test frameworks: The Penetration Testing Framework 0.59 [19], the Penetration Testing Execution Standard [20] and the NIST "Technical Guide to Information Security Testing and Assessment Regarding the attacking side" [26].

2.3.8.1.1 The Penetration Testing Framework 0.59

This framework is presented by the author as a "good starting resource for those getting into this (pen test) field".

It includes a template for all the following steps:

- Pre-inspection visit template (i.e. defining the scope of the project, retrieving the autorization, credentials,...).
- Network Footprinting (Reconnaissance).
- Discovery and Probing.
- Enumeration.
- Password cracking.
- Vulnerability assessment.
- Specific testing for Bluetooth, Wifi, Cisco, Citrix and VOIP services.
- Exploitation.
- Final report.

2.3.8.1.2 The Penetration Testing Execution Standard (PTES)

Regarding the second studied framework, as described on their website [20] :

"It is a new standard designed to provide both businesses and security service providers with a common language and scope for performing penetration testing (i.e. Security evaluations). It started early in 2009 following a discussion that sparked between some of the founding members over the value (or lack of) of penetration testing in the industry."

It is composed of 7 mains sections which, according to them, "cover everything related to a penetration test - from the initial communication and reasoning behind a pentest, through the intelligence gathering and threat modeling phases [...]" [20]:

1. Pre-engagement Interactions.
2. Intelligence Gathering.
3. Threat Modeling.
4. Vulnerability Analysis.
5. Exploitation.

6. Post Exploitation.

7. Reporting.

Lastly, as indicated on their website, this standard is backed by a group of information security practitioners from all areas of the industry such as financial institutions, services providers or security vendors.

2.3.8.1.3 NIST Technical Guide to Information Security Testing and Assessment Regarding the attacking side.

As the third and last framework, it is made by the National Institute for Standards and Technologies which they describe to be "responsible for developing standards and guidelines, including minimum requirements, for providing adequate information security for all agency operations and assets" [26].

As indicated in the framework, its purpose is to provide guidelines for organizations on planning and conducting technical information security testing and assessments, analyzing findings, and developing mitigation strategies. Additionally, it is intended for use by computer security staff and program managers, system and network administrators.

Finally, this framework is divided into 8 categories that are relatively identical to those of the first 2 frameworks.

2.3.8.1.4 Common elements

During our review of these 3 Frameworks, we went through each intelligence gathering, vulnerability assessment, vulnerability analysis, exploitation and post-exploitation steps. The pre-engagement and reporting steps are considered out of scope here.

Out of this review, we note specific elements to monitor in order to ensure that the pentester can use his tools/methodologies which are:

- List of open ports.
- List or running processes.
- List of installed packages.
- System version.
- List of users and their passwords.
- The needed files[14] to complete an exploitation.
- Network interfaces.
- Firewall state.
- Name of the device.

2.3.8.2 Defenders side

After determining the elements covered by an attackers, we studied the elements watched by the defender side.

[14] It could be a database content, a textfile, any files which need to be found by the pentester.

2.3.8.2.1 Indicator of compromise (IoC)

As defined on wikipedia [17] :

"Indicator of compromise in computer forensics is an artifact observed on a network or in an operating system that, with high confidence, indicates a computer intrusion.

Typical IoCs are virus signatures and IP addresses, MD5 hashes of malware files, or URLs or domain names of botnet command and control servers".

Although there are initiatives to standardise IoC, this has not yet been fully achieved. Some IoC creation/parsing tool such as iocbucket.com, iocextractor.com, and even a python library (ioc-parser) have been tested.

We found a lack of standardization when a .IoC file made by one solution could not be properly ridden/parsed by another one.

We concluded in not using this kind of file but studied the content.

2.3.8.2.2 Example of IoC

The .IoC file is usually following the XML format. In figure 2.6, we see an example of an such a file from iocbucket website.

Figure 2.6: Example of IoC from iocbucket.com [16]

A lot of elements, including the one listed in section 2.3.8.1.4 may be indicated inside

an IoC file[15], the complete list will not be developed as it is out of scoop for this research.

Finally, it should be noted that .ioc files are not used to monitor desired items, but to look for indications of a compromised system.

2.3.8.3 Conclusion

After watching the elements needed for an attacker and the elements monitored by a defender, we found that the number of elements watched by the first group is more limited than the second one. And all of these can be an indicated as a type of element in .ioc file.

In the end, we concluded that the elements necessary to monitor in order to ensure that a system is always in a state of possible corruption can be limited to the list of elements, defined in section 2.3.8.1.4, watched by an attacker.

[15]For example, for a given file, its date of creation, date of last modification, content, hash, path, ... Every possible information can be an IoC.

Chapter 3
Project's mission, objectives and requirements

In this section, we explain what we aim to bring with this project.

After doing so, we explain the objectives and develop each requirements that they involve.

Additionally, we talk about wanted features that are not required for the project but would be welcome.

Finally, we explain the elements that were covered or not by the state of the art, who this project is for, and its scoping.

3.1 Mission statement of this project

As stated during our state of the art in section 2.2.2, the popularity of cyber ranges and cyber exercise has been growing since 2000.

These exercises aim to improve the skills of cyber specialists or to test the security of a certain infrastructure by copying it to a cyber range.

In some cases, these exercises are hosted by a certain entity, but are led by a third party instructor who communicates directly with the participants and the organizers.

The instructor is not always one of the creators of the infrastructure (he is not a system administrator of the cyber range nor an owner who can modify the infrastructure). In most of these particular cases, the instructor has the solution of the exercise, the steps to follow and the needed credentials as well.

Unfortunately, these infrastructures are not often directly accessible to a third-party (such as an instructor and/or a participant). A technology like a VPN for example may be needed. And in some cases, it is necessary to go through a DMZ in order to access a wanted device.

When going through a given device to access another one, we call the first one "the pivot".

While it is relevant to require a pivot, it has the drawback of making monitoring more complex for a third party.

Indeed, during our state of the art of monitoring solutions of section 2.2.2, we did not find any allowing to monitor indirectly accessible machines without installing an agent or additional packages on the targeted machines. A solution without additional installation is necessary, because it is neither possible for the instructor to install an additional element each time or practical to do so. Take the example of a machine that has failed and has been reset by a system administrator, the instructor would have to reinstall his agent, and reconfigure everything in order to monitor it again.

In addition, as this solution would allow a third party to monitor the status of the system, it could also allow a participant to ensure that a machine is still accessible and/or has an unmodified state after manipulating it.

This is more efficient and practical than simply pinging it continuously, as it is sometimes done.

However, in the majority of UNIX environment, these machines are accessible via SSH or OpenVPN. We therefore assume and limit our work to devices accessible through SSH protocol as a first step.

Unlike the instructor, who has the solution and all the credentials, the participant will have to build the monitoring system machine by machine. Indeed, as he does not have all the credentials, he will not be able to monitor every machines at the beginning. But the more he progresses, the more information he will be able to get regarding the infrastructure.

At the end, such a solution would allow third-party to have the opportunity to quickly identify an error and report it to the administrators. This would save time for all parties involved.

3.2 Objectives

The project has three main objectives :

1. Finding a well suited monitoring solutions for an external user of a cyber range (such as an instructor and/or participant).

2. Determining the mains elements to monitor in order to ensure the "exploitable" state of the infrastructure.

3. Allowing the monitoring solution to reach all the needed devices (directly and non-directly accessible[1]).

To achieve the first objective, a review of the SotA of the monitoring solution for a cyber range has been done and the sota of popular free to use solutions as well.

The second objective was completed by defining the typical elements sought by the pentester during a cyber exercise and merging them with the elements monitored by the defenders of an infrastructure.

And finally, the third objective can be completed by combining an existing monitoring solution with SSH tunnels.

3.3 Requirements

In this section, we describe each requirements to respect the project objective.

[1] As explained in section 3

3.3.1 Able to monitor the needed information with an alert system

The aim is to ensure that the machines are still in the desired state (which could be a vulnerable one or not), as to maintain a good working infrastructure for participants and thus maximise the benefits of these when using the range.

Which means that, having an alert system on the monitored devices becomes a requirement.

We defined the elements to be monitored in the section 2.3.8.1.4.

Finally, we want the solution to use easy to configure and reusable configuration file as input.

3.3.2 Make as few modifications as needed on target machines

As the main objective is to monitor devices that are not owned by the user of the solution, The less we modify the machine the better it is. Indeed, we do not want to modify the stability of the infrastructure or to introduce an undesirable side effect in the cyber range.

Finally, as, this may not be possible in some cases, a solution which would require the target state modification (as installing an agent, additional packages, modifying the firewall rules...) would not satisfy our requirements.

3.3.3 Being able to reach all the machines within the range (directly or not directly accessible).

Additionally, we need to be able to easily monitor machines that are only accessible via a pivot. The devices can be behind a VPN or accessible by jumping through another device first (c.f. section 2.1.8).

This requirement allows a third-party (such as an instructor or participant of a cyber exercise, which are included in this category) that has no special rights on the infrastructure (as a direct access and creation/modification capabilities on the whole infrastructure) to monitor the desired machines.

3.4 Welcome features

Indeed, as the requirement represent the measurable elements to fulfil the project, this section covers the elements we would want that are not required for the success of the project, but would be welcome for its usage.

3.4.1 Using an existing monitoring tool

Using an existing monitoring tool has the advantage of not reinventing the wheel. It is not a feature per say but we find it to be an important point. As we saw in the state of the art, the monitoring tools available to the public are already very effective. They have configurable alert systems, configurable dashboards, and already approved ways of monitoring. Doing it all over again from scratch seems completely inefficient for us, and less likely to get used by someone as they need to learn a dedicated tool.

Indeed, by focusing on allowing these kind of solution to monitor a device behind a pivot, we have **(1)** a bigger possibility of a user being comfortable with using our tools, **(2)** no need to re-implement every features available in these popular tools, which have already

been tested and verified, and **(3)** a bigger chance of an user customizing and modifying our tools and thus extending our solution.

3.4.2 Having a Guest Dashboard

The last feature that would be welcome, if manageable, is the possibility of the user of our solution to share some information (not the sensible one as a target password or credentials but just the output of a monitoring task) with other people.

In a cyber exercise, this could allow, for example, two things: **(1)** the instructor to give hints to the participants, and **(2)** the participants not to have to instantiate the monitoring solution if the instructor can share it.

3.5 Requirements covered by state of the art

We started by searching the scientific literature for a monitoring solution that could be used by an instructor or participant in a cyber range (someone who is not on the infrastructure management team). And we found nothing of the sort (c.f. section 2.2.2).

So we took a look at the popular free-to-use tools such as ELK stack, Zabbix and Prometheus. During our study of these tools, we eliminated ELK stack and Prometheus and found an adequate one : Zabbix.

Zabbix is popular, has a monitoring tool via SSH, a customizable dashboard for the user and/or a guest and an alert system.

In conclusion, every requirements except *the ability to, monitor a non-directly accessible device without modifying its state* are covered by Zabbix.

3.6 Requirements not covered by state of the art

During our literature review, we have not found any studies related to a monitoring solution for a participant or external instructor regarding a cyber exercise.

In addition, we have found no solution allowing an agentless monitoring of a non-directly accessible device (in both the academics papers and the available commercial products).

3.7 Project scoping

The monitoring solutions available (in scientific research as well as commercial products) are already well developed. We're not focusing on improving any particular monitoring tool, but we're using a popular monitoring tool to solve a particular problem for a specific public.

3.7.1 Who is this project for ?

We focused on finding a monitoring solution for a third-party (such as an instructor and/or a participant of a cyber exercise). These categories of people differ from an owner or administrator of the infrastructure in that they do not have full rights and/or direct access to the infrastructure.

By providing them with a monitoring tool compatible with the target infrastructure, they will be able to detect and report a problem more quickly, resulting in a more efficient exercise with less time wasted on faulty targets.

3.7.2 What target is this project for ?

The objective of this project is to be able to monitor an infrastructure that is not owned by the person who monitors it.

This means that we have to be able to monitor elements on machines that are not directly accessible, and with a minimum of modification, because we cannot assume that it is always possible to modify them.

However, in most cases, these machines are accessible via SSH. We therefore assume that they all have an SSH server configured with a user account.

3.7.3 For what kind of exercise ?

We want to monitor the elements necessary for the proper functioning of a pen test type of exercise in particular.

This is the reason why we defined the elements to monitor as the one in section 2.3.8.1.4

3.7.4 What do we use ?

Each of the used technologies are defined in section 2.

To deploy a VM, we used Terraform combined with Microsoft Azure. To provision a machine, we used Ansible.

Regarding the monitoring solution, we chose Zabbix. And to allow Zabbix to perform agentless monitoring of non-directly accessible machines, we developed in Python our script for creating SSH tunnels.

3.7.5 Explicit out-of-scope definition

We have conducted all our experiences on a virtual cyber-ranges. We reasonably believe that our study should be relevant for physical and hybrid ranges as well. But for time budget reason, we did not test our solutions on these kinds of cyber ranges, we thus cannot ensure it works on them even though we think it should.

For budget reason as well, we only monitor the infrastructure state. The solution does not modify them when a change has occured[2]. Also, the monitoring solutions aim to ensure that the infrastructure is still in a desired state (which could be a vulnerable one),

Additionally, we focus on pen test exercises. This means that our monitoring solution may be less relevant or not relevant at all for other exercises that require monitoring of other elements.

Finally, we do not provide solutions for monitoring machines that are not accessible via SSH or OpenVPN by the user (with or without pivots). We leave as future work the support of other protocols for remote administration and/or pivoting.

[2]e.g. we are not implementing features to transform a "traditional" infrastructure in an immutable one. Which might be a desirable feature for certain use cases and will be left for future work.

Chapter 4
Experimentation & Implementation

In this chapter, we describe the experiments that led us to the implementation of our proposed solution, followed by the actual implementation of the solution.

The solution itself allows an external person of a cyber range to monitor the wanted elements on a targeted UNIX device.

To do so, we deploy a Zabbix server and create SSH tunnels in order to allows Zabbix to execute BASH command on a UNIX device. The output of the command is then retrieved and monitored.

Finally, Zabbix alerts the user when a monitoring task output has changed.

The source code and the whole documentation are available on the gitlab (c.f. section C.1). The main files are available in section B.

4.1 Experiments

In this section, we describe **(1)** why we chose Zabbix as the monitoring solution, **(2)** how we adapted it to monitor non-directly accessible devices, and finally **(3)** how we tried to make our solution more user-friendly.

4.1.1 Choosing the Monitoring solution

During our literature review, as described in section 2.2.2, we have not found a solution filling our needs described in section 3.

This is why we looked at popular monitoring solutions: Zabbix, ELK stack and Prometheus, which are explained in section 2.3.1 to see if we could adapt them to our needs.

4.1.1.1 Zabbix vs ELK stack vs Prometheus

The first step is to determine which one suits our needs defined in section 3. In order to do so, we read the documentation for each of these solutions to see their monitoring features.

We started with ELK stack, and looked at the way it collects data.

ELK stack collects its data, via BEATS agent, which is, as its name suggests, an agent to be installed on the target machines. We have not found any features, already made, that allow ELK stack to collect the data we are interested in without an agent.

But, considering the functioning of ELK stack, which is in several parts (one to retrieve the information in a certain standard, one to analyze them and the other to visualize them) it may be possible to create a script which will, in an agent-less way, retrieve the desired information and send it to the ELK stack server.

Before taking this way, we decided to explore Prometheus and Zabbix first.

In a second step, we looked at Prometheus. This second solution uses what they call "exporters"[1] to retrieve the desired metrics. These exporters are either developed and maintained by Prometheus, or they can be made and shared by the community.

[1] More info about them here: https://prometheus.io/docs/instrumenting/exporters/

For example, the "Apache exporter" allows an user to retrieve the metrics about the Apache server.

There are also more general purpose of exporter such as the "BlackBox exporter" which allows the probing of endpoints such as HTTPS, HTTP, TCP,... This exporter does not require to log in to the target for example, but probes the data from the outside (hence the "Blackbox" name).

Here, we are interested in exporters marked as "official" that are supported and developed by Prometheus. And among them, we have not found one which would suit our needs.

We have not gone through all the unofficial exporters because we do not want to rely on a module that may be obsolete/not maintained in the future or not validated by Prometheus themself.

Finally, before thinking about developing our own exporter we decided to explore the Zabbix possibility.

On the Zabbix side, we found its documentation to be the most clear and complete.

No need to search, one by one, for all the possible extensions as with Prometheus. And no need to read the documentation of several different tools as with ELK stack.

Zabbix is a all-in-one solution, meaning that you get all the features out of the box. And out of the box, Zabbix proposes several agents.

While these agents must all be installed, Zabbix also offers remote SSH command execution via what they call an "SSH check". As indicated on their website [46]:"SSH checks are performed as agent-less monitoring. Zabbix agent is not needed for SSH checks."

This Zabbix feature allows us to probe a device every X seconds in order to retrieve the output of a command executed through SSH.

In addition, Zabbix, as ELK stack and Prometheus, allows the creation of triggers to alert an user (trough the GUI, email, or other communication canals) when an output changes for example.

4.1.1.1.1 Resume

Out of the box, we did not find wanted features in ELK stack and Prometheus. For security and maintenance concern, we did not want to rely on non official "modules" which are not validated by the official product team.

Finally, by simply reading each solution features, we chose to go with Zabbix. Out of the box, it handles the execution of remote commands through SSH. This feature allows us to retrieve the needed information determined in section 2.3.8.1.4.

In addition, we can configure triggers on these output to alert us when an output follows a given rule (as when an output is different than expected, when it is different than the last one, when the value is below the one desired,...).

4.1.2 Agentless monitoring using Zabbix.

The first step is using Zabbix to monitor a wanted element on a given device.

To do so, we configured Zabbix to execute a command through SSH, retrieve the output, monitor it, and alert us when it changes or is different than what we expected.

The figure 4.1 shows a configuration of a task monitoring the `/etc/network/interfaces` of a device with the IP address 192.168.1.105.

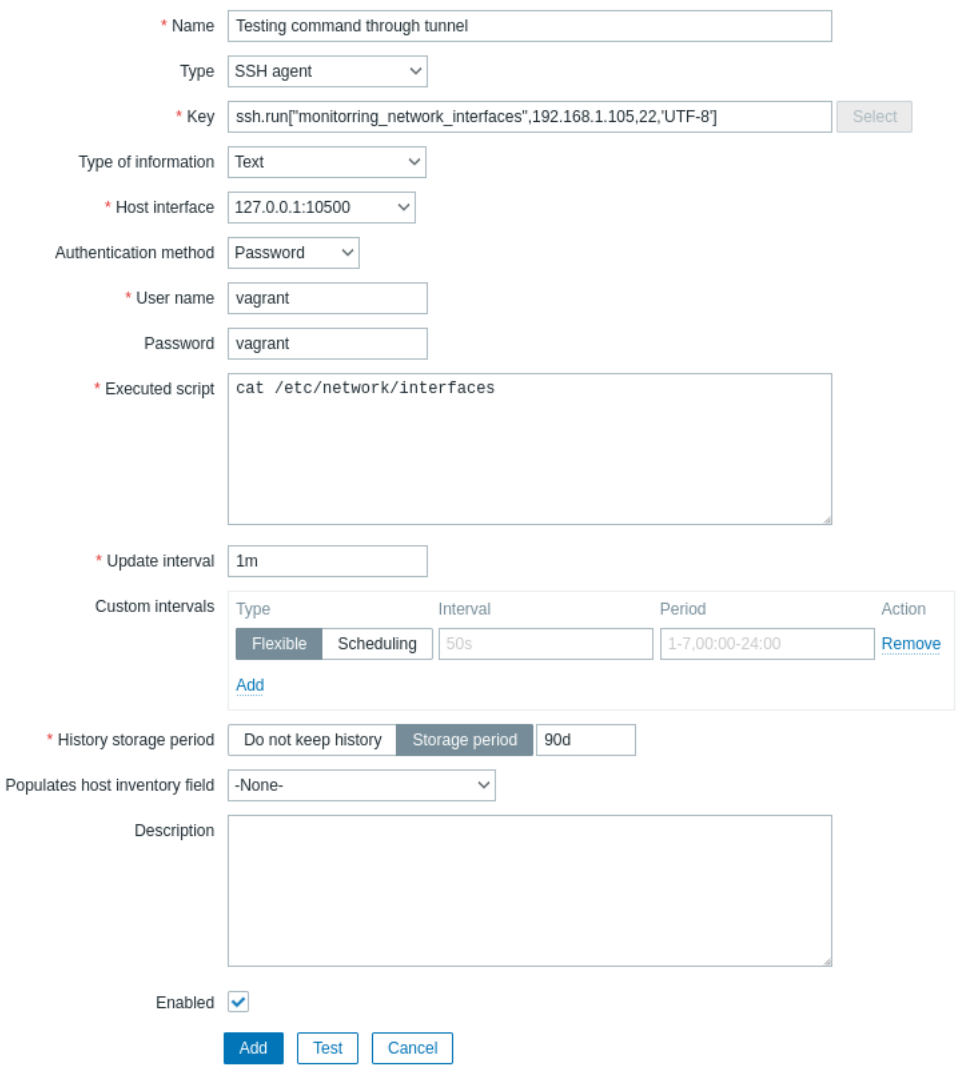

Figure 4.1: Illustration of an agent less monitoring task using Zabbix.

4.1.3 Monitoring devices behind a pivot

Now that we have a monitoring solution which allows us to execute commands through SSH and monitor its output, we're facing the first challenge which is to monitor a non directly accessible device.

4.1.3.1 SSH tunnels

An already proven solution to directly reach a device behind a pivot is through the usage of SSH tunnels.[2] We will not prove the validity of SSH tunnels[3], but we prove the integration of tunnels with Zabbix to monitor remote devices.

4.1.3.1.1 Basic case of a tunnel combined with Zabbix

In our case, in order to allow Zabbix to directly monitor via its SSH agent a machine behind a pivot, we establish a tunnel on the Zabbix server that links to the target machine.

[2] In this section, we will refer a "SSH tunnel" as simply a "tunnel".
[3] It is already a popular and proven solution

The figure 4.2 represents the scenario of an SSH tunnel deployed by the following command:

```
ssh -L ZabbixServerPort:Target1IPAddress:Target1Port pivotUsername@pivotAddress
```

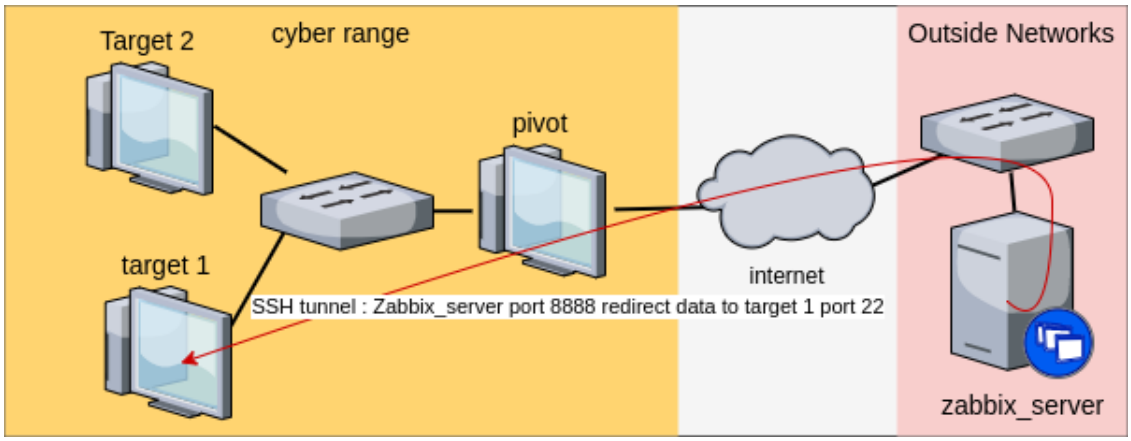

Figure 4.2: Illustration of a simple SSH tunnel in red.

With this solution, we don't modify the target state and we allow Zabbix to execute command through SSH on the Target1.

The picture 4.3 is a screenshot of the item configuration in the Zabbix GUI. Once the tunnel is deployed, we did select the "SSH agent" (which is the Zabbix SSH check with the execution of a command), and we did indicate the local address and the port on which the tunnels will redirect to the SSH port the Target1.

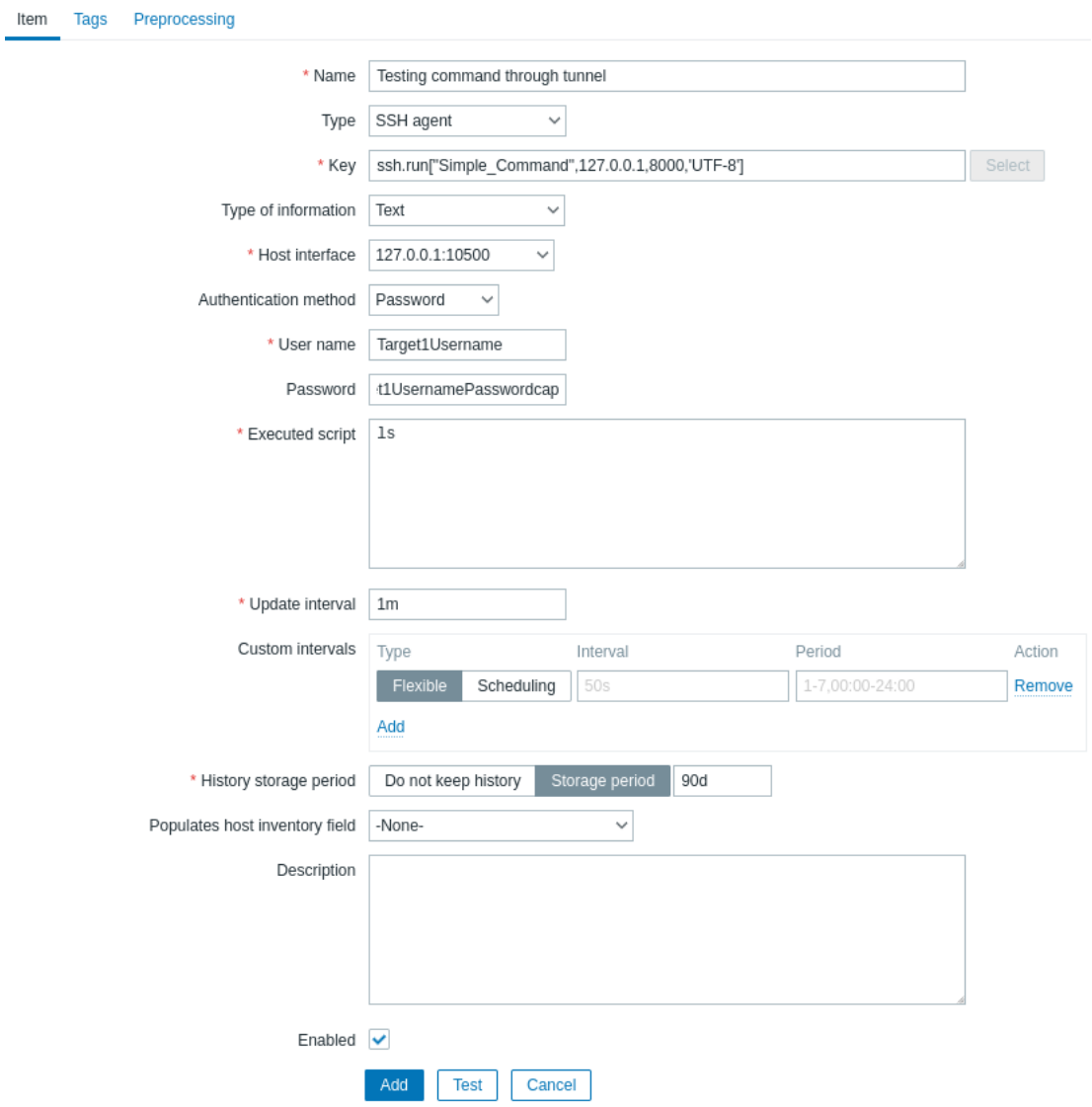

Figure 4.3: Illustration of a monitoring item creation through a SSH tunnel.

Finally, Zabbix allows the testing of the monitoring task to verify its configuration, and as we can see in picture 4.4, Zabbix is able to execute the command on the Target1.

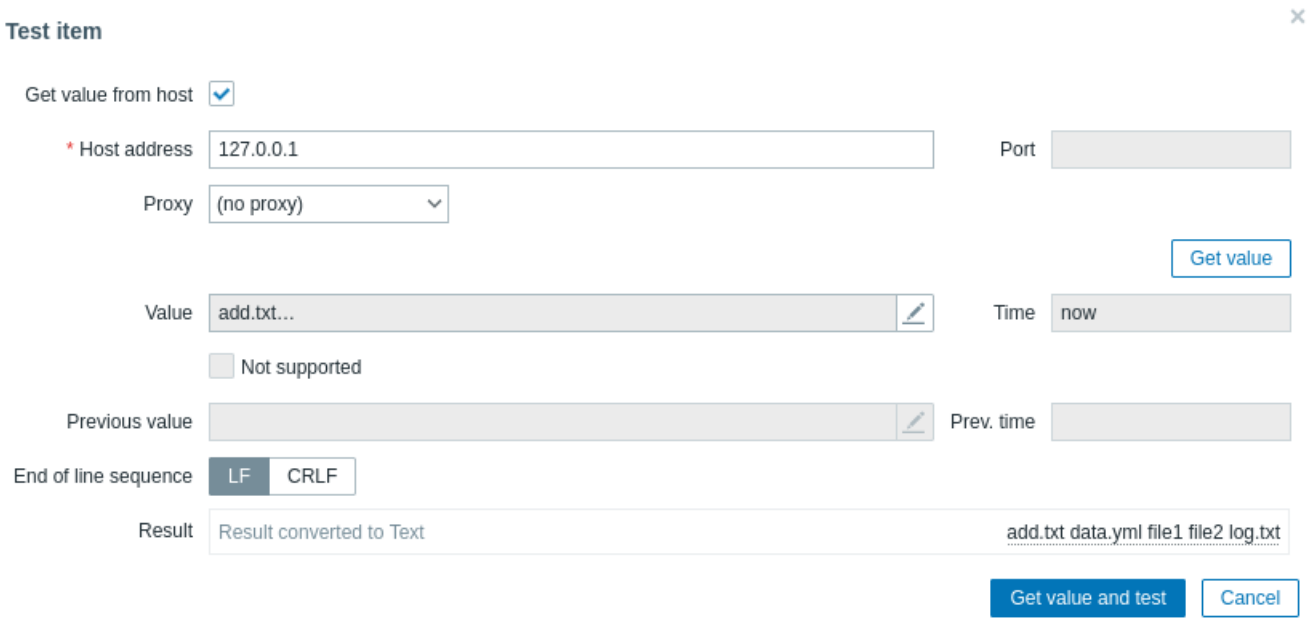

Figure 4.4: Testing the Zabbix task through the SSH tunnel.

4.1.3.1.2 Multi hop tunnel combined with Zabbix

No we have covered a basic example, but sometimes, more than one hop is required to reach a target.

Let's take the next illustration as an example.

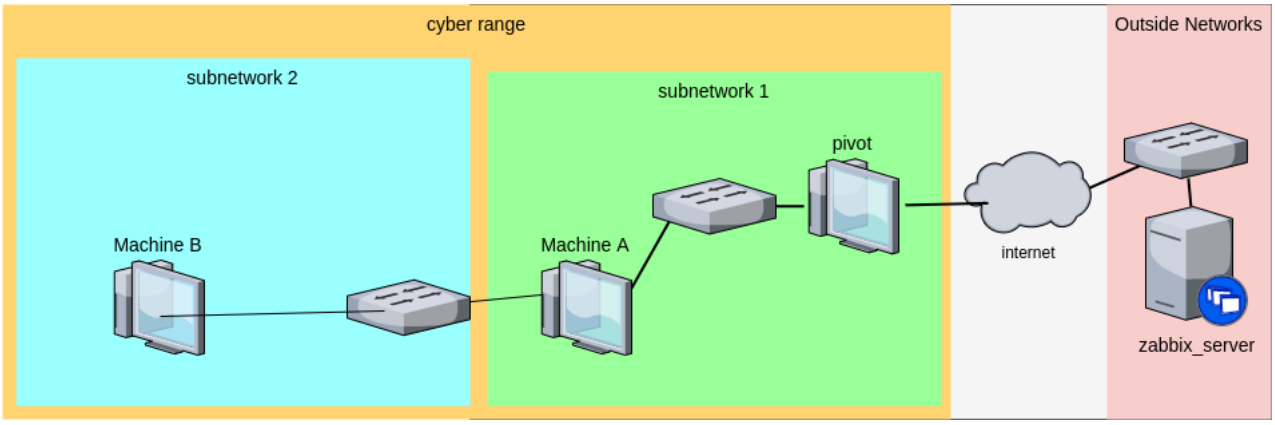

Figure 4.5: Illustration of a more complex infrastructure.

In the image above, we see that it is necessary to go through two hops to reach the Machine B. The two hops are the "pivot" and the "Machine A" devices.

For such a scenario, there are 2 possibles solutions.

(1) The first possible solution would be to create 2 tunnels, one between the **Zabbix server** and the **machine A** (tunnel 1), and the other tunnel between the **machine A** and **machine B** (tunnel 2).

This solution, as illustrated in figure 4.6, requires to "stick" the tunnels together (i.e. redirecting the exits of tunnel 1 to the entry of tunnel 2 to reach the Machine B). It has the advantage of being able to have single tunnels. This means that if tunnel 1 breaks

(connection problems or other), tunnel 2 would still be active. We would only need to redeploy tunnel 1.

But the disadvantage is that changes have to be done on machine A in order to create the second tunnel. Which is something we want to avoid.

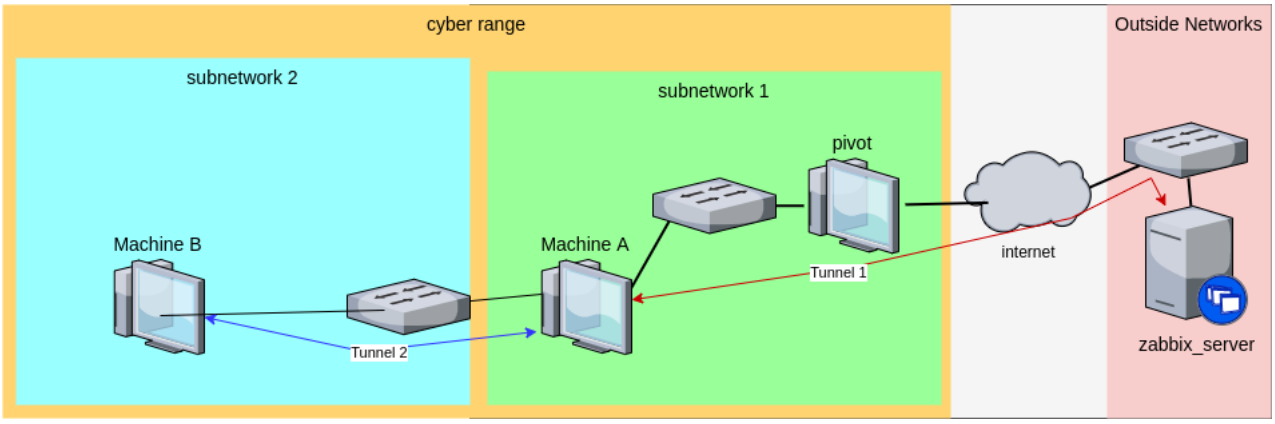

Figure 4.6: Illustration of a solution with 2 tunnels.

The other solution is to build the second tunnel inside the first one, as showed in figure 4.7.

This solution has the disadvantage of having to redeploy everything if a single link in the chain breaks, but the advantage of not changing the state of machine A.

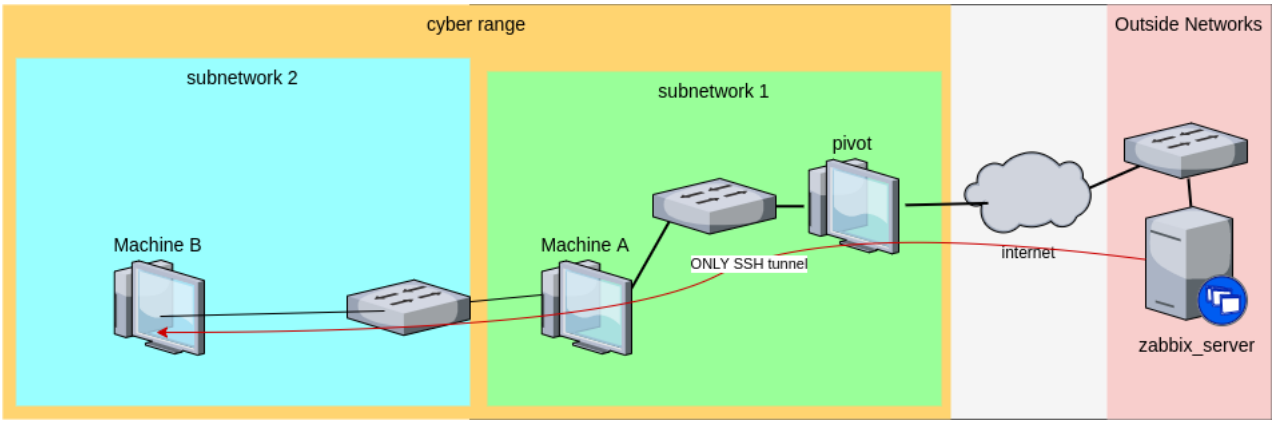

Figure 4.7: Illustration of a solution with 1 tunnel

The above tunnel is deployed using the *JumpHost* feature of the ssh command:

```
ssh -J X@PivotAddress,Y@machineAAddress \
Z@MachineBAddress -L localPort:localhost:PortMachineB
```

Although the second solution has the disadvantage of having to redeploy the entire tunnel when one of the links in the chain breaks, it is still preferable than the first solution requiring to modify the pivot state.

4.1.3.1.3 Single hops to access multiple devices

The tunnels deployment technique established in the last sections is suitable for the two previous points, but is not very effective when there are several machines to access behind the same pivot, in fact, a tunnel must be deployed for each device when the infrastructure follows the figure 4.8.

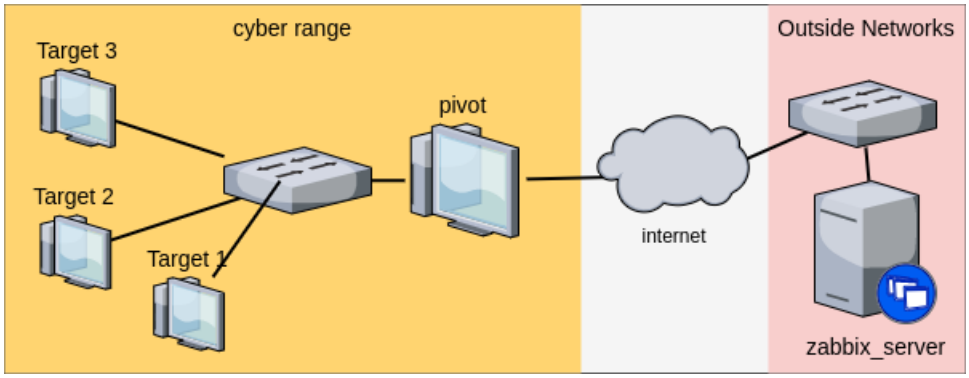

Figure 4.8: Multiples devices accessible behind one pivot.

Instead of deploying multiple tunnels in this case, we tried to use the pivot machine as a proxy for Zabbix using the ssh command :

```
ssh -D 8000 PivotUsername@PivotPassword
```

Unfortunately, Zabbix can be configured to work with a dynamic proxy if and only if a Zabbix proxy agent is installed on the pivot machine.

And since we want to avoid modifying the pivot, we stay with the solution requiring one tunnel per machine.

4.1.3.1.4 Accessing devices in a VPN

In order to allow Zabbix to monitor a device inside a VPN[4], we tested two ways to do it:

1. Configuring the VPN on the Zabbix server with configuration file generated for a client.

2. Deploying a tunnel from the Zabbix server using a device publicly accessible and connected to the VPN as a pivot.

The first way allowed Zabbix to monitor the devices inside the VPN, and if a tunnel is needed inside the VPN, it can be deployed as well.

The second way allowed Zabbix to monitor the devices behind a VPN without actually being connected to the VPN. A tunnel has to be deployed using a device reachable by Zabbix and connected to the VPN as a pivot.

In conclusion, both ways are possible and need to be configured by the user according to the possibility of connecting the Zabbix server directly to the VPN or not.

[4]During our tests, we used the OpenVPN package.

4.1.4 Conclusion of the tests

In this section, we have tested the usage of Zabbix to execute an agentless monitoring of a device.

After that, we used SSH tunnels to allow Zabbix to monitor non-directly accessible devices, and inside a VPN as well.

Now that we have the needed tools in order to make a solution filling our requirements in section 3, we are suggesting to automate the solution deployment in the next section.

4.2 Setup, Requirements, Environment & Tools

After testing different possibilities, and finding a solution that fills our needs, we want to automate each required steps for its deployment.

In this section, we describe the code[5] we wrote and the steps needed to have a Zabbix server, create the SSH tunnels and configuring the monitoring tasks.

4.2.1 List of requirements to use the solution

- Terraform (on the device used to deploy the Zabbix server).
- Python 3 with several packages (on the devices on which the tunnels are deployed).
  - pytunneling
  - pyyaml
  - sshtunnel
- Ansible (on the device used to deploy the Zabbix server).

4.2.2 Deploying a Zabbix server

For this step, we used the *Infrastructure as Code* tool: Terraform.

In addition, the Université Libre de Bruxelles has a partnership with Microsoft Azure, which gives each student a user account with available credits.

So we used Terraform to deploy a Debian 11 machine on the Azure cloud. On which we then installed and configured our Zabbix server.

We chose this method because **(1)** Infrastructure as code is becoming more and more popular and widely used and **(2)** because, as the configuration of Zabbix has been automated through an ansible-playbook, a user can deploy a virtual machine on his device and then run the script in order to have a Zabbix ready to be used in a few steps.

4.2.3 Creating the SSH tunnels and the monitoring tasks

We wanted to make the solution simple to use, and to do so, we regrouped all the parameters inside a yaml file. This type of file has the advantage of being human readable and easy to parse as well.

[5]The whole code with a complete README is available on the gitlab: https://gitlab.com/Amuranov/automated-monitoring-system-for-cyber-range

Inside the configuration file, we provide every devices information (IP address, username, password), the wanted tunnels (the local port to run on, the hops to make) and all the elements to monitor.

The tunnel creation script takes the needed parameter to create the wanted tunnels.

The monitoring task creation take the elements to monitor on the wanted devices and replace their IP address with the tunnels coordinates when needed.

As we chose to make an easy to write configuration file, we needed to implement our own parser in order to retrieve the wanted information, replacing the IP with the tunnels,... And to do so, we decided to implement the scripts using Python3 as we were to limited by Ansible features.

We can see an example of configuration in figure 4.9. This configuration file is to monitor several point on 2 devices. The machine2 device is reachable by using the First_tunnel which uses machine1 as a pivot.

```yaml
hosts:
  machine1:
    ssh_address_or_host: 192.168.1.2
    ssh_username: machine1User
    ssh_password: machine1Password
  machine2:
    ssh_address_or_host: 10.10.2.2
    ssh_username: Machine2User
    ssh_password: Machine2Passwoord

tunnels:
  - tunnel_name: First_tunnel
    path:
      - machine1
      - machine2
    local_bind_address:
      address: "0.0.0.0"
      port: 9000

monitoring:
  zabbix_server:
    api_address: http://192.168.1.58/zabbix/
    username: ZabbixUsername
    password: ZabbixPassword
  targets:
    machine1:
      port: 22
      command:
        - systemctl status apache2
        - ip a
      files_content:
        - /etc/passwd
        - /home/user1/.ssh/id_rsa
    machine2:
      tunnel: First_tunnel
      files_content:
        - /etc/network/interfaces
      open_ports:
      users:
      command:
        - ls
```

Figure 4.9: Example of a configuration file with 2 devices and 1 tunnel.

Lastly, as we have determined the main elements that need to be monitored for the users, we implemented some "modules" in python. These "modules", as we can see on figure 4.9 are specified by the user in the configuration file. For example, when the user want to monitor the content of a file, he specifies "files_content" followed by the files he wants to monitor, instead of having to specify the whole bash command to execute on the monitored devices.

In addition, like in Ansible, we implemented a "command" module. It allows the user, if he is limited by our modules, to provide a bash command and Zabbix will monitor its output.

Finally, to deploy the tunnel on a remote machine if needed (in our case, as we had a

Zabbix server in the cloud, we deployed the tunnels on it), we wrote an ansible-playbook allowing the installation of requirements and the deployment of the tunnels on the remote server. It can be run every time one needs to redeploy the tunnels. This playbook and its inventory are available in section B.3.

4.3 Results

At the end, after deploying the Zabbix server, creating the tunnels, and pushing the monitoring tasks on the server, we had a web page displaying all the chosen elements to monitor.

The user can then use every Zabbix features as filtering the data in order to visualize all the elements for a given machine for example.

Additionally, Zabbix alerts when a machine is not reachable, when an output has changed,... If the user needs more features, he can configure them using Zabbix GUI.

Nonetheless, we automated the steps which we found necessary (creating and alert when a device is not reachable, and another one when an output for a given task changes). Finally, a screenshot of this page is available on figure B.3.

Chapter 5

Experiment's output Analysis

In this chapter, we describe the key performance indicators (K.P.I.) for each requirements defined earlier in section 3, and the tests we made to ensure that these requirements are met.

After that, we will discuss the welcome features and finish by a conclusion of the solution output.

5.1 Test infrastructure

In order to run our tests, we have manually deployed an infrastructure according to the following image.

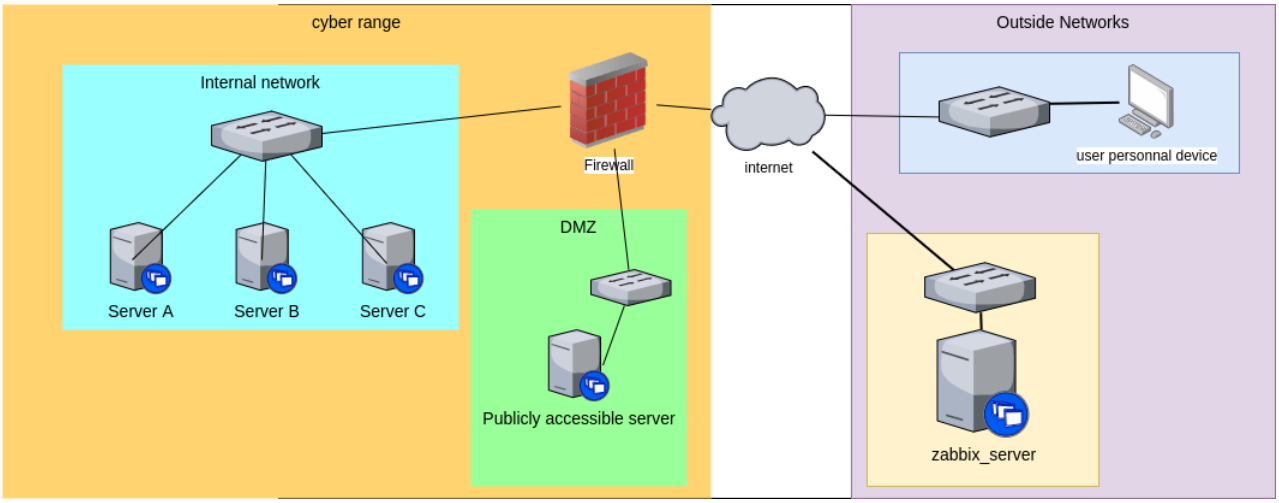

Figure 5.1: Topology of our testing infrastructure.

Within the infrastructure, we implemented a cyber range which includes:

- A DMZ (to simulate a typical infrastructure requiring a pivot to access the local network devices).
- A firewall.
- An Internal network with 3 devices only reachable from within the DMZ through SSH protocol.

Every devices, except the **user personnal device** were running on Azure B2s[1] virtual machine with Debian 11 running on them. The **user personnal device** was a Desktop computer with an i5 3500k and 32 Gb of ram running on Ubuntu.

The whole infrastructure was running on relatively modest specs.

[1]It is a popular virtual machine running on 2 virtual processor with 4Gb of ram, costing 35 USD per month(= 0.05 USD per hour).

In addition to this, we deployed a Zabbix server on a different network, and the whole infrastructure was managed by a pc outside the network as well.

5.2 Verification of our requirements

5.2.1 Being able to reach all the machines within the range (directly or not directly accessible)

Before verifying the monitoring capabilities or how our solution affects the target, we have to ensure its ability to reach any wanted device.

5.2.1.1 K.P.I

For this requirement to be met, we want our solution to be able to :

- Reach any wanted directly accessible device.
- Reach any devices that require one pivot before being accessible (by providing the path).
- Reach any devices that require more than one pivot before being accessible (by providing the path).
- Reach a device within a VPN.

5.2.1.2 Basic case

The first step is to test the correct integration of SSH tunnels with Zabbix SSH agent features.

We created three tunnels using our script (by providing the **Publicly accessible server** and the **Server A, B, and C** credentials).

By doing that, the Zabbix server was indeed able to reach the devices running on the Internal network.

The representation of the topology was similar to the next figure (with the tunnels in blue, red and yellow).

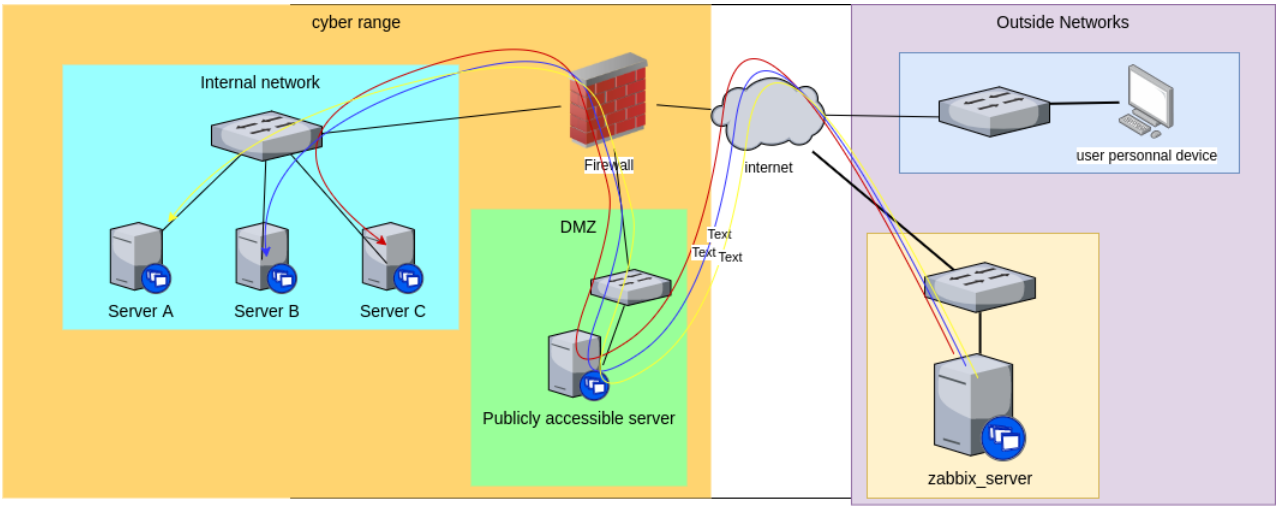

Figure 5.2: Topology of the infrastructure with the tunnels.

5.2.1.2.1 How to reproduce this step

In order to deploy the tunnels on the zabbix server, we ran the following command from the user PC: `ansible-playbook main.yml -i inventory`.

This playbook[2] ensures that every needed packages to deploy the tunnels are installed on the target (the zabbix_server in our case), then upload the python3 script `tunnel_creation.py` before deploying the tunnels.

There were 2 configuration files for this step[3].

(1) The ansible inventory (which contains the zabbix_server information):

```
---
all:
  hosts:
    zabbix_server:
      ansible_host: 20.224.111.156
      ansible_python_interpreter: /usr/bin/python3
      ansible_user: simpleuser
      ansible_ssh_pass: "Ahg14!H?sdA"
      ansible_become_method: sudo
      ansible_become: "yes"
      ansible_become_pass: "Ahg14!H?sdA"
      tunnel_script_destination: "/home/simpleuser/"
```

(2) The `data.yml` configuration file[4], in which we indicated the credentials of Server A, Server B, Server C, the publicly accessible server and the tunnels to build as well:

```
hosts:
  dmz:
    ssh_address_or_host: 13.93.24.55
    ssh_username: dmzuser
    ssh_password: dhGa78:!aop
  serverA:
    ssh_address_or_host: 10.0.0.2
    ssh_username: userservera
    ssh_password: ksdhr!pzer7
  serverB:
    ssh_address_or_host: 10.0.0.3
    ssh_username: userserverb
    ssh_password: 45Aehfd!jq,
  serverC:
    ssh_address_or_host: 10.0.0.4
    ssh_username: userserverc
    ssh_password: ,sdghq4YuaB
```

[2]Which is available and more detailed on the gitlab: https://gitlab.com/Amuranov/automated-monitoring-system-for-cyber-range

[3]The deployed devices, for which you see the IP and password, have been deleted afterwards.

[4]Which is used by the python3 tunnel_creation.py file

```
tunnels:
  - tunnel_name: to_serverA
    path:
      - dmz
      - serverA
    local_bind_address:
      address: "0.0.0.0"
      port: 8889
  - tunnel_name: to_serverB
    path:
      - dmz
      - serverB
    local_bind_address:
      address: "0.0.0.0"
      port: 8890
  - tunnel_name: to_serverC
    path:
      - dmz
      - serverC
    local_bind_address:
      address: "0.0.0.0"
      port: 8891
```

After that, we logged in to the zabbix_server and were able to SSH into the server A, server B and Server C by running, respectively:

```
ssh userservera@127.0.0.1 -p 8889
ssh userserverb@127.0.0.1 -p 8890
ssh userserverc@127.0.0.1 -p 8891
```

Finally, the Zabbix server was able to monitor these devices as well.

5.2.1.3 Multi-hop case

As a second test, in order to test the multi-hop capacity, we deployed another internal network which was only reachable from within the first internal one through the firewall.

Secondly, within the second internal network, we connected the server D to another network which is composed of the server G. As shown on figure 5.3, the server D is the only one able to access the Server G.

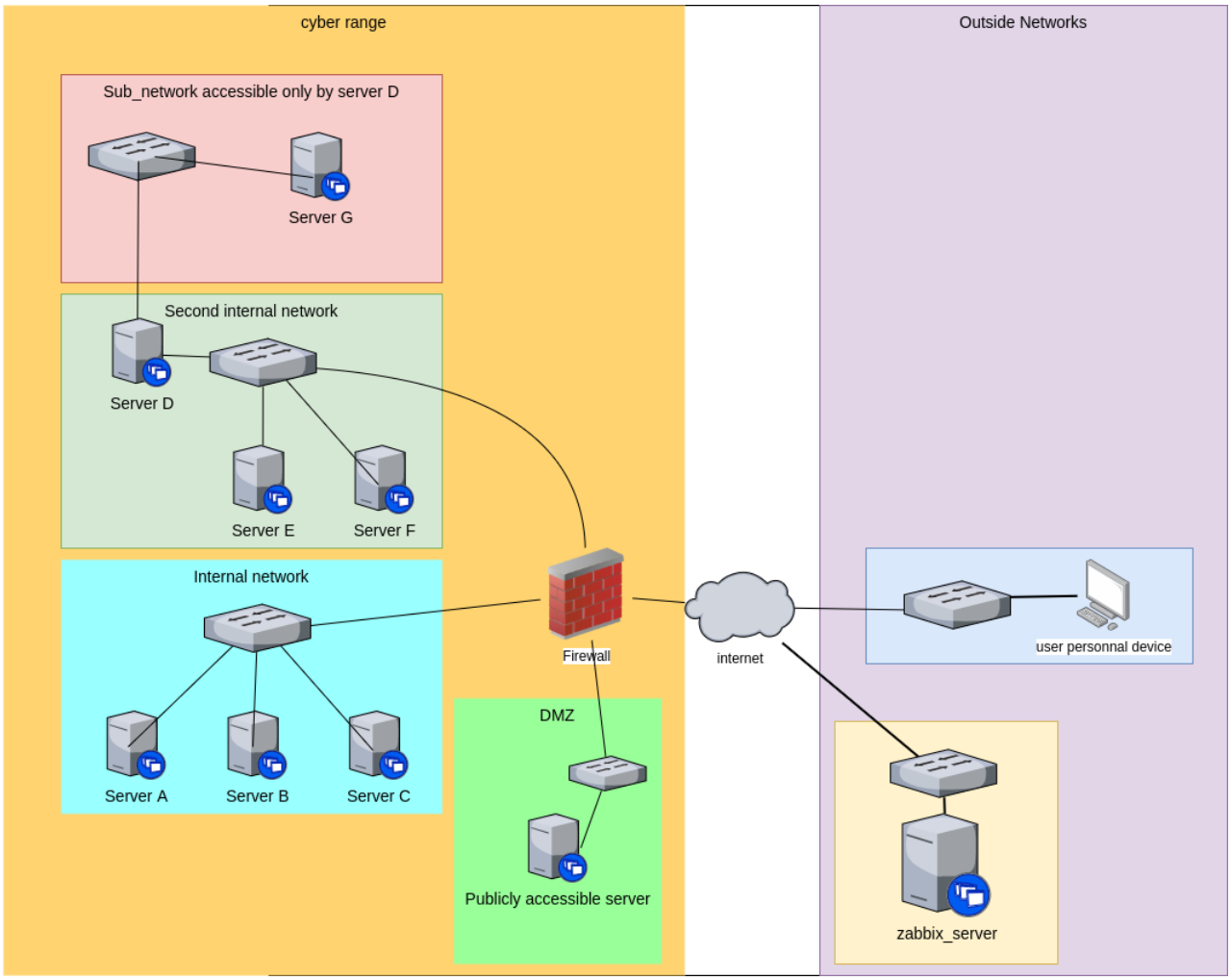

Figure 5.3: Topology of a multi-hop infrastructure.

To allow the Zabbix server to reach the server D, server E and server F we deployed three tunnels using the **publicly accessible server** as first pivot, then the **Server A** as second pivot[5].

Lastly, we deployed a forth tunnel using the **publicly accessible server** as first pivot, the **Server A** as second pivot and the **Server D** as a third one to access Server G.

Once again, the zabbix_server was able to connect to the server D, E, F and G.

5.2.1.3.1 How to reproduce this step

For this test, we executed the same code as the basic case of section 5.2.1.2.1. Only the tunnel configuration file was changed. The following one was used:

```
hosts:
  dmz:
    ssh_address_or_host: 13.93.24.55
    ssh_username: dmzuser
    ssh_password: dhGa78:!aop
```

[5] As all the devices within the "internal network" were able to access any of the devices within the "second internal network", we could have used the Server B or server C as a second pivot as well.

```
  serverA:
    ssh_address_or_host: 10.0.0.2
    ssh_username: userservera
    ssh_password: ksdhr!pzer7
  serverB:
    ssh_address_or_host: 10.0.0.3
    ssh_username: userserverb
    ssh_password: 45Aehfd!jq,
  serverC:
    ssh_address_or_host: 10.0.0.4
    ssh_username: userserverc
    ssh_password: ,sdghq4YuaB
  serverD:
    ssh_address_or_host: 10.0.1.2
    ssh_username: userserverd
    ssh_password: kjqgf4!at1Z
  serverE:
    ssh_address_or_host: 10.0.1.3
    ssh_username: userservere
    ssh_password: Jdh,az41!ha4
  serverF:
    ssh_address_or_host: 10.0.1.4
    ssh_username: userserverf
    ssh_password: Had4,97!2dq
  serverG:
    ssh_address_or_host: 10.0.2.2
    ssh_username: userserverg
    ssh_password: 15Azhq!,7ep


tunnels:
  - tunnel_name: to_ServerA
    path:
      - dmz
      - serverA
    local_bind_address:
      address: "0.0.0.0"
      port: 8889
  - tunnel_name: to_ServerB
    path:
      - dmz
      - serverB
    local_bind_address:
      address: "0.0.0.0"
      port: 8890
  - tunnel_name: to_ServerC
    path:
```

```
    - dmz
    - serverC
  local_bind_address:
    address: "0.0.0.0"
    port: 8891
- tunnel_name: to_ServerD
  path:
    - dmz
    - serverA
    - serverD
  local_bind_address:
    address: "0.0.0.0"
    port: 8892
- tunnel_name: to_ServerE
  path:
    - dmz
    - serverA
    - serverE
  local_bind_address:
    address: "0.0.0.0"
    port: 8893
- tunnel_name: to_ServerF
  path:
    - dmz
    - serverA
    - serverF
  local_bind_address:
    address: "0.0.0.0"
    port: 8894
- tunnel_name: to_ServerG
  path:
    - dmz
    - serverA
    - serverD
    - serverG
  local_bind_address:
    address: "0.0.0.0"
    port: 8895
```

5.2.1.4 Hops within a VPN

As a last test for this section, we took the infrastructure of figure 5.1 again, and tested 2 points with the VPN.

(1) The first point was to use the Zabbix server to monitor the whole infrastructure after configuring the VPN (using a client .opvn file generated by the VPN server) directly on it. When it is possible to connect the Zabbix server directly to the private network, it is able to monitor des whole infrastructure. The usage of tunnels inside the VPN is possible

too[6].

(2) The second point was to verify the ability to hop through a host reachable by Zabbix and with the VPN configured on it. We were able to use the host as a pivot to reach devices inside the VPN as well.

5.2.1.5 Conclusion

After running our basic case test, and complex case test, we concluded that our K.P.I. have been reached.

Our solution is able to reach **(1)** a directly accessible device, **(2)** one that requires a pivot, **(3)** one that requires several one and **(4)** one that is inside a VPN with one or several pivot.

5.2.2 Able to monitor the needed information with an alert system.

The second requirement is the ability of our solution to monitor the elements defined in section 3.

In addition, the solution must raise an alert as soon as a change in any of these elements is noticed.

5.2.2.1 K.P.I.

To consider the second requirement as met, the solution needs to be able to:

- Monitor the elements defined in section 2.3.8.1.4.
- Monitor the elements via SSH connection.
- Raise an alarm when a monitored element has changed.
- Easily specify in a reusable format for each machine what to monitor and have the monitoring and alerting system automatically deployed/configured based on that single reusable input.

The last indicator allows the user to efficiently deploy/redeploy the solution.

5.2.2.2 Monitoring the wanted elements.

During the integration test of SSH tunnels made in prior section, we tested the monitoring of the elements defined earlier in section 2.3.8.1.4.

First, we configured the Zabbix server using our script, `zabbix_api_communication.py`, in order to automate the monitoring task creations. The infrastructure used for this step was the one used earlier in figure 5.3.

The `zabbix_api_communication.py` script takes the same file as input as the `tunnel_creation.py` as it needs to know how to reach a device (through a tunnel, or not).

Additionally, this script needs information indicating what to monitor on what device. To do so, we used the same `data.yml` file as in section 5.2.1.3.1, and we added the following block:

[6] In the case where a pivot is needed even after connecting to the VPN.

```
monitoring:
  zabbix_server:
    api_address: https://20.224.111.156/zabbix/
    username: Admin
    password: Uopwb!az.fg,
  targets:
    dmz:
      port: 22
      command:
        - systemctl status ufw
    serverA:
      tunnel: to_ServerA
      network_interfaces:
    serverB:
      tunnel: to_ServerB
      file_content:
        - /etc/passwd
        - /var/log/auth.log
    serverE:
      tunnel: to_ServerE
      users:
      version_info:
    serverD:
      tunnel: to_ServerD
      network_interfaces:
      command:
        - systemctl status ufw
    serverG:
      tunnel: to_ServerG
      network_interfaces:
      users:
      version_info:
      open_ports:
      command:
        - apt list --installed
```

This script pushes the configured tasks to the Zabbix server and replace the address of the host with the tunnel entry address when it is specified for a device.

A screenshot of the output is available in section B.5.

Each time, we were able to monitor the following elements through our created modules:

- The list of users.
- The list of open ports.
- The list of running processes.
- The content of a file.

- The network configuration.
- The list of installed packages.
- The version information.
- The Name of the device.
- Any additional information[7] using the **command** module.

5.2.2.3 Alerting system.

We configured the Zabbix server to alert the user as soon as an output differs from the wanted one.

To do so, we assume the infrastructure is in the wanted state when Zabbix monitors the device for the first time. Then, we used the Zabbix trigger features[8] to raise an alarm when an output of a monitoring task is different than the first output ever recorded by the system.

Indeed, Zabbix allows the users to implement their own trigger settings by providing "Testing expressions". In our case, we compare the first element of a task, with the last one, and when they differ, an alarm is raised.

For example, the trigger expression for the network interfaces monitoring of the publicly accessible server used as example in section 5.2.2.2 is:

```
last(ssh.run[dmz_network_interfaces,13.93.24.55,22,'UTF-8'],#1)<>
first(ssh.run[dmz_network_interfaces,13.93.24.55,22,'UTF-8'],100000000)=1
Enabled
```

As soon as an output of a monitoring task differs from the first one, the user sees a red alert on its main interface. Such an alert can be seen, in the annex on section B.4.

Additionally, The alert can be set up to send a mail, a text message, or messaging using a Social Network like Discord for example.

5.2.2.4 Conclusion

During the testing of our solution monitoring capabilities, we condluded that our K.P.I. have been reached.

The solution is indeed able to monitor, **(1)** via SSH, **(2)** the needed elements, **(3)** with an alarm system and using **(4)** an easy to specify and reusable format.

5.2.3 Make as few modifications as needed on target machines.

5.2.3.1 K.P.I.

And for the last requirement, the solution needs to:

- Have the less possible impact on the device during its runtime.
- Make no persistent changes on the devices (which are either monitored or used as a pivot).

[7] not yet implemented as a module

[8] More information available at: https://www.zabbix.com/documentation/current/fr/manual/config/triggers/trigger

To verify these requirements, we made two tests to ensure that our solution makes the less possible changes on a given device when it is monitored and/or used as a hop for a tunnel. The goal of the first test is to check for the impact on devices at runtime, and the other one to check for persistent changes on the devices.

To do so, we deployed a fresh Debian 11 virtual machine with the minimum packages installation.

5.2.3.2 Runtime

To see the impact of being monitored and used as a hop in an SSH tunnel, we deployed a second Zabbix server.

The second Zabbix server was used to monitor the Debian 11 VM before, while, and after it was monitored by the first Zabbix server. Through the SSH log files, we noticed that Zabbix does, indeed, simply connect to the device, execute a command and then close the session.

For example, this is what we noticed, just after the Zabbix server executed a monitoring task through SSH, in the `/var/log/auth.log` of the monitored Debian device:

```
April 5 16:30:15 debian-11 sshd[23710]:
Accepted password for basicuser from 20.224.111.156 port 52966 ssh2
April 5 16:30:15 debian-11 sshd[23710]:
pam_unix(sshd:session): session opened for user basicuser by (uid=0)
April 5 16:30:15 debiansystemd-logind[1209]:
New session 805 of user basicuser.
April 5 16:30:15 debian-11 sshd[23710]: pam_unix(sshd:session):
session closed for user basicuser
```

Additionally, we used our solution to monitor the target state while it was used as a hop for a SSH tunnel.

We noticed no modification in the list of open ports or installed packages.

However, while the device was used as a hop in an SSH tunnel, we noticed a new process. The created process looked like a simple SSH connection from a remote user to the device.

5.2.3.3 Persistence

To verify that our solution does not change the state of the system, we looked at every file on that system and compared their hashes before/after the tunnel deployment.

However, as some files are modified during the simple functioning of the machine[9], we captured the state of the machine at five different times T[10] using the following bash commands[11]:

```
sudo find /home/ -type f -exec md5sum {} \; > /home/user/home_1.txt && \
sudo find /media/ -type f -exec md5sum {} \; > /home/user/media_1.txt && \
sudo find /root/ -type f -exec md5sum {} \; > /home/user/root_1.txt && \
```

[9] such as those in the root /proc folder for example

[10] each time was spaced about 10min apart, just enough to generate all the hashes

[11] We ran this script five times, and changed the output file number at each of these times.

```
sudo find /sys/ -type f -exec md5sum {} \; > /home/user/sys_1.txt && \
sudo find /var/ -type f -exec md5sum {} \; > /home/user/var_1.txt && \
sudo find /boot/ -type f -exec md5sum {} \; > /home/user/boot_1.txt && \
sudo find /mnt/ -type f -exec md5sum {} \; > /home/user/mnt_1.txt && \
sudo find /run/ -type f -exec md5sum {} \; > /home/user/run_1.txt && \
sudo find /tmp/ -type f -exec md5sum {} \; > /home/user/tmp_1.txt && \
sudo find /dev/ -type f -exec md5sum {} \; > /home/user/dev_1.txt && \
sudo find /opt/ -type f -exec md5sum {} \; > /home/user/opt_1.txt && \
sudo find /usr/ -type f -exec md5sum {} \; > /home/user/usr_1.txt && \
sudo find /etc/ -type f -exec md5sum {} \; > /home/user/etc_1.txt && \
sudo find /proc/ -type f -exec md5sum {} \; > /home/user/proc_1.txt && \
sudo find /srv/ -type f -exec md5sum {} \; > /home/user/srv_1.txt && \
sudo find /bin/ -type f -exec md5sum {} \; > /home/user/bin_1.txt && \
sudo find /sbin/ -type f -exec md5sum {} \; > /home/user/sbin_1.txt && \
sudo find /lib32/ -type f -exec md5sum {} \; > /home/user/lib32_1.txt && \
sudo find /lib64/ -type f -exec md5sum {} \; > /home/user/lib64_1.txt && \
sudo find /lib/ -type f -exec md5sum {} \; > /home/user/lib_1.txt && \
sudo find /libx32/ -type f -exec md5sum {} \; > /home/user/libx32_1.txt
```

Between the times T1;T2 and T4;T5, we did nothing on the machine (apart from hash generation) in order to capture the evolution of the files during a simple run of the machine. During T3, we deployed the tunnels.

So if a file is different between times T2;T3 but the hash is also different between times T1;T2 and T4;T5, we thought it was reasonable, but we may be wrong, to think that it has changed due to the machine running and not because of our solution.

After generating all the hashes, we wrote and ran a python[12] script to look for a difference between the hashes of a file at the different T intervals.

In the following example, we can see the script used to analyse the differences of the files in the /home directory:

```
file1= "home_1.txt"
file2= "home_2.txt"
file3= "home_3.txt"
file4= "home_4.txt"
file5= "home_5.txt"

list1 = []
list2 = []
list3 = []
list4 = []
list5 = []

with open(file1, "r") as f1:
    for line in f1:
        list1.append(line)
```

[12]On which we did not spend a lot of time.

```
with open(file2, "r") as f2:
    for line in f2:
        list2.append(line)

with open(file3, "r") as f3:
    for line in f3:
        list3.append(line)

with open(file4, "r") as f4:
    for line in f4:
        list4.append(line)

with open(file5, "r") as f5:
    for line in f5:
        list5.append(line)


for k in range(len(list1)):
    if list1[k] == list2[k] == list3[k] == list4[k] == list5[k]:
        print("No differences found for element " + k)
else :
        print(list1[k])
        print(list2[k])
        print(list3[k])
        print(list4[k])
        print(list5[k])
    print("========")
```

When the python script noticed a file that had not the same hash every time, it printed all his hashes. By doing that, we were able to see at what moment the file was different, if it was at every T moment, or just after running the monitoring solution.

Regarding the results, we found new files at each T moment in the `/home` directory. Which was normal, as it simply was the newly created files containing the hashes every time.

The hashes of every files in the following directories were the same at the five T moments:

```
/tmp
/dev
/opt
/usr
/etc
/srv
/bin
/sbin
/lib32
```

```
/lib64
/lib
/libx32
/media
/mnt
/boot
```

In the following directories, we found the hashes of several files being different at each T moment:

```
/var
/sys
/root (which contains the history of the executed bash commands,
      as we ran the script as root)
/run
/proc
```

Finally, in these directories, we did not find a file that had a different hash only between the moment T2 and T3 (before and during the execution of the solution) or between the moment T3 and T4 (during and after the execution of our monitoring solution).

Given our way to test this point and how limited in time and budget we were, we could be wrong, but we consider our test good enough for the verification of this point.

Indeed, if we needed to be more precise and had the time to do it, we could have analysed the contents of each file with different hashes at each point in time individually to see more precisely what caused the change, and whether our solution was the cause or not .

5.2.3.4 Conclusion

In conclusion, we determined that a simple SSH process is created on a device when it is used as a hop for a SSH tunnel.

In addition, we did not find any information indicating a persistent change of our monitoring solution on a given device when is it monitored and/or when it is used an hop within an SSH tunnel.

Finally, our solution does not make persistent changes on a device, but leaves detectable prints behind (in the `/var/log/auth.log` file for example).

5.3 Additional features

In addition to the requirements, our solution proposes a welcome feature which can be automatically configured in the future as well.

5.3.1 Guest Dashboard

Out of the box, Zabbix allows a "guest login" which gives access to a dashboard containing limited information such as the number of hosts to be monitored, if there is an alert that has been activated and any wanted information. Such a user can connect by clicking on "sign in as guest" as shown on the following screenshot:

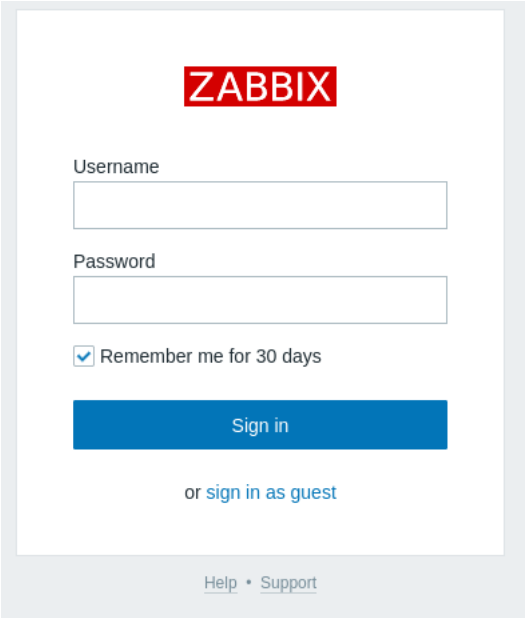

Figure 5.4: Zabbix login page.

After being logged in, the guest is redirected to the same Zabbix interface but with limited privileges. For example, he can not initially access sensitive data as a target address or credentials, but the rights can be modified by the owner of the Zabbix server.

This feature allows, for example, a user to set up the monitoring system and to share it with the others. It might save other users to instantiate and use their own monitoring system.

However, we did no automate the configuration of the dashboard for time budget reason. We leave that as a future contribution.

5.4 Conclusion

In this section, we have analyzed our solution by checking if every initially declared requirements have been fulfilled and have concluded that it is the case.

The discussion on what needs to be improved, or comments on the how the requirements are met is done later in section 7.

Chapter 6
CyberSecurity analysis of the proposed solution

Now that we have implemented a solution and verified its validity, we can analysis its security.

There are different approaches that we can take; such as a static/dynamic code analysis, an architecture review, a penetration test, ...

In our case, we chose to take an audit approach by analyzing the coverage of activities, i.e. the compliance in this case, with the NIST framework [29] defined in section A.1.

We made this choice for two reasons.

(1) The first reason was the will by the researcher to learn to analyse the compliance with such a framework. This means that this analysis should be taken with caution, as the researcher is learning, there may be errors in some sections.

(2) We thought that analysing compliance with NIST (and leaving sections blank where necessary) could save time for any entity wishing to use our solution within their own organization.

We decided to let the other approaches as a future work.

Unfortunately, we greatly underestimated the time needed to make the project NIST compliant.

We have made a first draft available in the appendix with a table summarising the compliant, non-compliant, to be determined later points at the end.

Out of the 43 points of the NIST framework we studied, in the first draft, we are not compliant with 18 of them, and could not answer in advance for 15 other points (because they are elements to be defined by the organisation using our solution, which we cannot specify in advance, such as the points ID.AM-4, ID.BE, ID.GV, ID.RA,...)

6.1 Conclusion

To summarise, we have preferred to leave the first draft in the appendix, rather than establishing a messy cyber security report.

It was ambitious of us to compare our relatively small project with the NIST framwork within such a small time frame.

Thus, as long as the analysis is not further developed, we consider that no security analysis has been done for this project at the moment. We leave this part as future work.

Chapter 7
Discussion

In this chapter, we discuss about the difference of our research with the one explained during the SotA in section 2.

Finally, we explain the limitation of our solution and how it can be improved in the future.

7.1 Comparison with state of the art/related works

During our state of the art, we found plenty of researches about the monitoring of an infrastructure as developed in section 2.2.1. In most cases, the researched focused on monitoring a specific part of the cyber range (as we did), but they all made it from the infrastructure owner point of view.

These solutions required special rights on the infrastructure, such as a direct access on every devices or the rights to install additional packages allowing the monitoring of these devices.

However, we focused on a solution that would allow a third party (such as a participant or instructor in a cyber exercise) to monitor the elements they need to. Even for an infrastructure to which they do not have direct accesses.

In our case, we used Zabbix, as another researcher did to monitor its whole cyber range (c.f. "Using an emulation testbed for operational cyber security exercises. Springer, Heidelberg" [1]), but we adapted it in order to monitor the targets from outside the infrastructure. We chose Zabbix because of its ability to execute and monitor the execution of commands via SSH (vs its agent based monitoring feature used in the other research, which required admin rights on the infrastructure) which we combined with SSH tunnels in order to access non-directly accessible devices.

Indeed, where other solutions allow a more complete monitoring or simply other elements, we have found a solution that allows a person who has no rights on the infrastructure (but has the credentials of the machines) with no direct access to monitor the desired elements of it.

In addition to this, we used the IaC paradigm to deploy our solution. And by doing this, we allow the users of our solution to be able to configure their systems and then deploy them with a minimum of effort. Once the configuration is set up, the user has a full graphical interface provided by Zabbix, which is already a popular and proven solution, that they can manipulate to modify the solution if they wish.

The advantage of using popular software is that a user will be more likely to use our solution, as they are more likely to already be familiar with it. Furthermore, when the user is more inclined to use our solution, there is a greater chance that the user will adapt and improve it. This makes the project more interesting in the long run.

Finally, as our solution uses IaC, and we have automated the tasks that we feel are necessary (such as deploying the virtual machine, configuring it as a Zabbix server, and then creating tasks and SSH tunnels), the user can automatically deploy our solution

with several commands (no manual configuration except for the configuration files to be modified). An example of the configuration file is available in section 5.

7.2 Lessons learned

If the project was to be done again, we would change the cybersecurity analysis approach of it.

We chose to take an audit approach and to analyse the compliance of our solution with the NIST framework. This task could not have been correctly done in the given time budget, as explained in section 6.

We completely underestimated the time needed to make our project NIST compliant (not to mention the points we cannot indicate in advance because it depends on the person, entity or organisation using our solution).

Finally, if we were to do it again, we would probably choose to perform a crystal clear pentest on our solution. This is a task that we, now, feel is more relevant and more easily achievable in the context of this project.

7.3 Limitations of validity

Regarding the validity of our solution, for reasons of time budget, we have only tested it on UNIX systems. And more precisely the Ubuntu and Debian based distributions.

Furthermore, as our monitoring solution communicates with the machines via the SSH protocol, it is necessary that the machines we wish to monitor have a SSH server configured, reachable and accessible (credential wise) to be useful for anyone using it to monitor events.

And finally, when creating a tunnel, each hop must have the SSH server running on port 22. We have not, for the time being, left the possibility of specifying another port for these devices.

7.4 Future work

Concerning future work, the first possible contribution to the project would be to extend it, by making it work with Windows systems via Windows powershell. This would extent the validity of our solution. Especially since the use of Windows machines in a cyber range is not a singular event.

Secondly, modifying the modules to make them more modular would also be useful. Indeed, for the moment, certain parameters of a monitoring task are chosen by default.

For example, each monitoring task is performed every thirty seconds. It would be good to be able to give the user the choice of modifying the time parameter if they wish so.

The same applies to alerts when the output of a task changes. For the time being, we have set a high alert for each monitoring tasks. It would be nice to be able to either change the type of alert or to determine which alerts are more alarming than others through a survey of the target audience.

Thirdly, there are 2 points that could be improved regarding the tunnels creation.

(1) At the moment, when the user specifies the hops that the tunnel should take, we assume that each hop has the SSH server on port 22. This is sometimes not the case. So it would be good to allow the user, via the configuration file, to specify the port number on which the SSH server runs on each machine.

(2) In addition, making the tunnels more efficient would be a welcome upgrade as well. For the moment, we are creating a tunnel per machine to be accessed because we have not succeeded in combining Zabbix with dynamic forwarding (SSH -D, as explained in section 4.1.3.1.3) without modifying the state of the machine used as a pivot.

Fourthly, it would be useful to automate the configuration of the guest dashboard. At the moment, it is possible to use it but the user has to configure it manually via its graphical interface.

Automating this task would make the user more likely to deploy a guest page for other users.

Additionally, with this feature, we could consider to add a "fog of war" concept to a cyber exercise. We could implement a platform on which the user enters the flags captured during the exercise. Once a machine's flags are entered, this would trigger an automatic configuration of the Zabbix guest dashboard (configured by the instructor for example) to give access (to the user who entered the flag) to the monitoring information for the machine he has already captured the flag. The more flags the participant gets, the more accesses to devices he would have.

Fifthly, adding network auto discovery combined with the dynamic tunnel creation would be pertinent as well. For now, we manually describe the hops to take when deploying each tunnel. In the future, we could consider a solution to which we only give every devices credentials, and it dynamically finds the path to take to access to each one of them by looking at their routing table for example.

And lastly, we could think of a case where the user has more rights on the infrastructure (such as the right to modify the state of a machine for example) when using our solution. This case could be possible when our solution is used by the owners of the infrastructure (so it is not the case for which we have made our study here, but it is a possibility).

Indeed, if we have the rights on the infrastructure, the monitoring solution could capture a whole target state, and when its state is modified, the solution could restore it to the expected one. It could be done using Zabbix trigger features to execute a script which could restore the machine state for example.

This would, when possible and when there is a problem, save time for participants/instructors and administrators.

Chapter 8
Conclusions

For this master thesis, we implemented a monitoring solution allowing a person without a direct access to an infrastructure and with no special rights on it (such as a participant or an instructor of a cyber exercise) to monitor its devices.

In order to do so, we used an existing popular monitoring tool, Zabbix, and combined its agentless monitoring features with SSH tunnels to allow the user to monitor an infrastructure for which he has no direct access (as when a VPN is required, or jumping through an intermediate machine to access another one) without modifying it. When the monitored element of the device changes, the user automatically gets an alert on its Zabbix webpage.

The only requirement is to have a SSH user credential of the devices.

Starting with **Chapter 1**, we explain how such a tool could benefit a participant or an instructor of a cyber exercise.

We then explain the context of this research followed by a first definition of our problem.

We ended this chapter by describing the structure of this document.

After that, in **Chapter 2**, we defined the key concepts to understand this research, such as a cyber range, a monitoring software, a pivot, a SSH tunnel, a DMZ...

Following the definitions, we have reviewed the SotA regarding the cyber ranges and their monitoring solution.

During the SotA, we found no research focusing on such a problem : allowing a non-owner of an infrastructure with not special rights to it to monitor its state, and to know when an element has changed.

So, we looked at available free-to-use monitoring solution. Out of the box, we have not found a solution to our problem, so we looked at three popular free-to-use solutions and compared their listed features. Additionally, we determined the elements that needed to be monitored in our case (a cyber exercise from the participant/instructor point of view) by analysing the elements typically checked in this situation.

Following the SotA, in **Chapter 3**, we explained how the mission of this project is to allow a non-owner of a cyber range (such as a participant or an instructor, versus the infrastructure owner/creator) to monitor a device state to ensure it has not changed.

In this chapter, we then defined the technical requirements such as **(1)** having an alert system using an easy to configure and reusable format file, **(2)** the fewest possible modification on the devices and **(3)** the ability to reach every devices (directly or not directly accessible).

Then, we discuss how having a guest dashboard is a welcome feature as well and finally, we discuss how the requirements are or are not covered by the SotA.

Later, in **Chapter 4**, we described the experiments we made with Zabbix agentless monitoring feature and how we combined it with SSH tunnels.

After experimenting a solution, we automated its deployment with IaC paradigm using Terraform to deploy a machine and Ansible to provision the device into a Zabbix server.

Finally, we used Python3 to automate the creation of SSH tunnels and the creation of monitoring tasks on Zabbix by communicating with its API.

In **Chapter 5**, we explained the test infrastructure we deployed in order to try and analyze our proposed solution experimented in **Chapter 4**. We explained how the requirements are met, and if the additional features were filled as well.

Once we have experimented a solution and concluded that it filled our needs, we tried to realise a security audit on it in **Chapter 6**.

For this task, we followed the NIST framework: Identify, Protect, Detect, Respond and Recover.

This audit made us modify our solution to be more compliant, but as explained in the section, we were not able to be fully NIST compliant within the time budget.

At the end, in **Chapter 7**, we compared our solution with the SotA by explaining its differences with the other solutions found earlier. We explained how we focused on the non-owner of an infrastructure point-of-view, opposed to the others papers all focusing from the owner/administrator/creator point of view. Following this, we explained how our solution only works for UNIX devices at the moment.

Finally, we offered several possible future contributions such as making the solution work with Windows powershell, automating the configuration of the guest dashboard, or making the configuration of the monitoring task and SSH tunnels more flexible.

Bibliography

[1] A, P.G., C, S., M, M.: Using an emulation testbed for operational cyber security exercises. Springer, Heidelberg (Germany) (2017) [Cited on pages 10, 15, and 53.]

[2] Alfieri, R., Barbera, R., Belluomo, P., Cavalli, A., Cecchini, R., Chierici, A., Ciaschini, V., Dell'Agnello, L., Donno, F., Ferro, E., Forte, A., Gaido, L., Ghiselli, A., Gianoli, A., Italiano, A., Lusso, S., Luvisetto, M., Mastroserio, P., Mazzucato, M., Mura, D., Reale, M., Salconi, L., Sava, G., Serra, M., Spataro, F., Taurino, F., Tortone, G., Vaccarossa, L., Verlato, M., Vita Finzi, G.: The infn-grid testbed. Future Generation Computer Systems 21(2), 249–258 (2005), advanced Grid Technologies [Cited on pages 9 and 102.]

[3] Alvarenga, I.D., Duarte, O.C.M.B.: Rio: A denial of service experimentation platform in a future internet testbed. In: 2016 7th International Conference on the Network of the Future (NOF). pp. 1–5 (2016) [Cited on pages 9 and 102.]

[4] Braghin, C., Cimato, S., Damiani, E., Frati, F., Riccobene, E., Astaneh, S.: Towards the Monitoring and Evaluation of Trainees' Activities in Cyber Ranges, pp. 79–91 (11 2020) [Cited on page 10.]

[5] Brattstrom, M., Morreale, P.: Scalable agentless cloud network monitoring. In: 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud). pp. 171–176 (2017) [Cited on page 10.]

[6] Bulut, M.F., Sun, H., Arora, P., Vukovic, M., Koenig, K., Young, J.: Technical health check for cloud service providers. ArXiv abs/1906.11607 (2019) [Cited on page 11.]

[7] Čeleda, P., Čegan, J., Vykopal, J., Tovarňák, D., et al.: Kypo–a platform for cyber defence exercises. M&S Support to Operational Tasks Including War Gaming, Logistics, Cyber Defence. NATO Science and Technology Organization (2015) [Cited on pages 9 and 102.]

[8] Chandra, Y., Mishra, P.: Design of cyber warfare testbed (12 2015) [Cited on page 10.]

[9] Chandra, Y., Mishra, P.K.: Design of cyber warfare testbed. In: Hoda, M.N., Chauhan, N., Quadri, S.M.K., Srivastava, P.R. (eds.) Software Engineering. pp. 249–256. Springer Singapore, Singapore (2019) [Cited on pages 9 and 102.]

[10] elastic: Elk stack. `https://www.elastic.co/what-is/elk-stack` (2021), [Online; accessed 2022-01-01] [Cited on page 13.]

[11] European Union stats: Digital economy and society statistics - enterprises. `https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Digital_economy_and_society_statistics_-_enterprises` (2021), [Online; accessed 2022-01-01] [Cited on page 1.]

[12] forbes: More alarming cybersecurity stats for 2021 ! `https://www.forbes.com/sites/chuckbrooks/2021/10/24/more-alarming-cybersecurity-stats-for-2021-/?sh=7840ad9e4a36` (2021), [Online; accessed 2022-01-01] [Cited on page 1.]

[13] Hashicorp: Terraform. https://www.terraform.io/ (2021), [Online; accessed 2022-01-01] [Cited on page 15.]

[14] Herold, N., Wachs, M., Dorfhuber, M., Rudolf, C., Liebald, S., Carle, G.: Achieving reproducible network environments with insalata. In: Tuncer, D., Koch, R., Badonnel, R., Stiller, B. (eds.) Security of Networks and Services in an All-Connected World. pp. 30–44. Springer International Publishing, Cham (2017) [Cited on pages 9 and 102.]

[15] HG insights: Prometheus. https://discovery.hgdata.com/product/prometheus (2021), [Online; accessed 2022-01-01] [Cited on page 12.]

[16] Indicator of compromise: Indicator of compromise. https://iocbucket.com/iocs/02f60f2bb3c72c76b18eaf6326db18c6e71f24bb/download?1648053506000 (2021), [Online; accessed 2022-01-01] [Cited on pages IX and 18.]

[17] Indicator of compromise: Indicator of compromise. https://en.wikipedia.org/wiki/Indicator_of_compromise (2021), [Online; accessed 2022-01-01] [Cited on page 18.]

[18] Interpol: Interpol report shows alarming rate of cyberattacks during covid-19. https://www.interpol.int/News-and-Events/News/2020/INTERPOL-report-shows-alarming-rate-of-cyberattacks-during-COVID-19 (2021), [Online; accessed 2021-11] [Cited on page 99.]

[19] Kevin orrey: Penetration testing framework 0.59. http://www.vulnerabilityassessment.co.uk/Penetration%20Test.html (2021), [Online; accessed 2022-01-01] [Cited on page 16.]

[20] Kevin orrey: pentest-standard.org. http://www.pentest-standard.org/index.php/Main_Page (2021), [Online; accessed 2022-01-01] [Cited on pages 16 and 103.]

[21] Lee, S., Jung, S., Koo, H., Na, J.H., Yoon, H.Y., Shim, M.K., Park, J., Kim, J.H., Lee, S., Pomper, M.G., Kwon, I.C., Ahn, C.H., Kim, K.: Nano-sized metabolic precursors for heterogeneous tumor-targeting strategy using bioorthogonal click chemistry in vivo. Biomaterials 148, 1–15 (2017) [Cited on pages 9 and 102.]

[22] Mallouhi, M., Al-Nashif, Y., Cox, D., Chadaga, T., Hariri, S.: A testbed for analyzing security of scada control systems (tasscs). In: ISGT 2011. pp. 1–7 (2011) [Cited on pages 9 and 102.]

[23] Matildha Sjöstedt, R.M.: Monitoring of cyber security exercise environments in cyber ranges. p. 72 (2021) [Cited on pages 9, 10, and 11.]

[24] Muhammad Mudassar Yamin, Basel Katt, V.G.: Cyber ranges and security testbeds: Scenarios, functions, tools and architecture (2020-01), https://www.sciencedirect.com/science/article/pii/S0167404819301804, [Online; last access 2021-12-11] [Cited on pages 1, 9, 11, 12, and 102.]

[25] National cyber security center: Jump in cyber attacks during covid-19 confinement. https://www.swissinfo.ch/eng/jump-in-cyber-attacks-during-covid-19-confinement/45818794 (2021), [Online; accessed 2021-11] [Cited on page 99.]

[26] National Institut of standards and technologies: Special publication 800-115 technical guide to information security testing and assessment. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf (2021), [Online; accessed 2022-01-01] [Cited on pages 16 and 17.]

[27] NIST: cyber ranges. https://www.nist.gov/system/files/documents/2018/02/13/cyber_ranges.pdf (2018), [Online; accessed 2021-11] [Cited on page 4.]

[28] NIST: The five functions - cybersecurity framework. https://www.ansible.com/overview/how-ansible-works?hsLang=en-us (2021), [Online; accessed 2022-01-01] [Cited on page 70.]

[29] NIST: Framework for improving critical infrastructure cybersecurity. https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf (2021), [Online; accessed 2022-01-01] [Cited on pages 52, 62, 64, 65, 67, 69, 70, and 71.]

[30] NIST Agency: Cyber ranges. https://www.nist.gov/system/files/documents/2018/02/13/cyber_ranges.pdf] (2021) [Cited on page 9.]

[31] NIST Agency: Demilitarized zone. https://csrc.nist.gov/glossary/term/demilitarized_zone] (2021) [Cited on page 6.]

[32] NIST Agency: Penetration testing. https://csrc.nist.gov/glossary/term/penetration_testing] (2021) [Cited on page 4.]

[33] Prometheus: Prometheus. https://prometheus.io/docs/introduction/overview/ (2021), [Online; accessed 2022-01-01] [Cited on pages IX, 12, and 13.]

[34] Prometheus: Prometheus, from metrics to insights. https://prometheus.io/ (2021), [Online; accessed 2022-01-01] [Cited on pages 5 and 12.]

[35] RedHat: What is ansible. https://www.nist.gov/cyberframework/online-learning/five-functions (2021), [Online; accessed 2022-01-01] [Cited on page 15.]

[36] Shabeer Ahmad, Nicolo Maunero, P.p.: EVA: A Hybrid Cyber Range. ceurs-ws.org 1(1), 1–10 (2020) [Cited on pages 9 and 102.]

[37] Siboni, S., Shabtai, A., Tippenhauer, N.O., Lee, J., Elovici, Y.: Advanced security testbed framework for wearable iot devices. ACM Trans. Internet Technol. 16(4) (dec 2016) [Cited on pages 9 and 102.]

[38] SSh security: Ssh tunnel. https://www.ssh.com/academy/ssh/tunneling (2021), [Online; accessed 2022-01-01] [Cited on page 7.]

[39] Techopedia: Monitoring software. https://www.techopedia.com/definition/4313/monitoring-software (2021), [Online; accessed 2022-01-01] [Cited on page 5.]

[40] Tsai, P.W., Yang, C.S.: Testbed@twisc: A network security experiment platform. International Journal of Communication Systems 31, e3446 (10 2017) [Cited on pages 9 and 102.]

[41] Wikipedia: Virtual private network. https://en.wikipedia.org/wiki/Virtual_private_network (2021), [Online; accessed 2022-01-01] [Cited on page 5.]

[42] William Aubrey Labuschagne, M.G.: Developing a capability to classify technical skill levels within a cyber range. 16th European Conference on Cyber Warfare and Security ECCWS 2017 (2017) [Cited on pages 9 and 102.]

[43] zabbix: The elastic stack. `https://www.elastic.co/en/elastic-stack//)` (2021), [Online; accessed 2022-01-01] [Cited on page 12.]

[44] Zabbix: What is zabbix. `https://www.zabbix.com/documentation/6.0/en/manual/introduction/about` (2021), [Online; accessed 2022-01-01] [Cited on page 14.]

[45] zabbix: Zabbix monitoring solution. `https://www.zabbix.com/)` (2021), [Online; accessed 2022-01-01] [Cited on page 12.]

[46] zabbix: Zabbix ssh check. `https://www.zabbix.com/documentation/5.4/en/manual/config/items/itemtypes/ssh_checks)` (2021), [Online; accessed 2022-01-01] [Cited on page 26.]

Appendix A

CyberSecurity analysis of the proposed solution

A.1 NIST framework

The NIST framework is a set of guidelines and good practices to organize and improve our cyber security program.

As shown in figure A.1, the NIST framework is split into five main sections.

Figure A.1: NIST framework 5 functions.

Additionally, a colour coded summary table is available at the end of the chapter. A point with a green colour is compliant with the framework, a red colour indicates non-compliance, and a grey colour indicates a point not studied for our project because we found it either not relevant for our case, or we could not define this point at the moment.

A.1.1 Identify (ID)

The first functions aim to determine what are the assets and processes that need protection. Every description of a key point are available in the NIST framework [29].

A.1.1.1 Asset Management (ID.AM)

As explained in the framework, the section is about "the data, personnel, devices, systems, and facilities that enable the organization to achieve business purposes are identified and managed consistent with their relative importance to organizational objectives and the organization's risk strategy" [29].

A.1.1.1.1 ID.AM-1: Physical devices and systems within the organization are inventoried.

For the well functioning of the project, only one device is needed: the Zabbix server.

Finally, this server is intended to monitor a certain infrastructure which should be in the inventory as well but which can not be determined at this stage (as the aim of the solution is to be able to monitor any wanted infrastructure following the prerequisites).

A.1.1.1.2 ID.AM-2: Software platforms and applications within the organization are inventoried.

The following processes are needed on the Zabbix server to allow our solution to run:

- zabbix_server (to run the Zabbix software).
- zabbix_agentd (to communicate with monitored devices).
- apache2 (for the web interface).
- MariaDB (to allow Zabbix to store the outputs of the monitoring tasks).
- python3 tunnel_creation.py (this process handles the SSH tunnels).
- SSH server (to allow connection to the server).

A.1.1.1.3 ID.AM-3: Organizational communication and data flows are mapped.

We did not make such a data flow. Our solution is composed of a Zabbix solution which retrieves the information of the monitored targets and store them in its own database.

These information can be retrieved by the users of the solution, using its GUI (via HTTPS protocol), which is mainly the owner of the Zabbix server and the guests users if the owner configured them an access to these data.

Once the server is deleted, the data are deleted as well.

A.1.1.1.4 ID.AM-4: External information systems are catalogued.

Only external systems used are the ranges our system is connected to or deployed on which is a variable of our deliverable. Therefore, we can not and should not quantified this point.

A.1.1.1.5 ID.AM-5: Resources are prioritized based on their classification, criticality, and business value.

We have only one resource (the Zabbix server), which has the highest criticality and priority if limited to our deliverable. If we consider our proposal as well as the ranges it would be connected to, typically, considering the case study we initially designed it for (external monitoring of cyber ranges), we would consider our system to have a low-mid business value.

After all, the range should work independently of our system. Our objective is to enhance, facilitate the management/visibility of such exercise from a "customer"'s point of view (the instructor) but not substitute for proper practices to maintain the infrastructure from the owner.

A.1.1.1.6 ID.AM-6: Cybersecurity roles and responsibilities for the entire workforce and third-party stakeholders are established.

We can either consider the main use case for which our system was designed for, i.e. a single instructor deploying this tool to monitor an external solution, in which case that instructor would inherit all the responsibilities or, if this tool is integrated as part of a larger organisation (since it could be interesting for a team of pentesters as well as for internal monitoring), in which case determining these responsibilities would depend on the rest of the organization and therefore should be left to that organization and cannot be determined at this point.

A.1.1.2 Business Environment (ID.BE)

definition

"The organization's mission, objectives, stakeholders, and activities are understood and prioritized; this information is used to inform cybersecurity roles, responsibilities, and risk management decisions." [29]

Our solution can be used by any organisation, person or company wishing to use it in order to allow an external user to monitor a given cyber range. We leave this point to be filled in the future by the user as we can not specify the organization mission, objectives, stakeholders or activities in advance for every user of our solution.

A.1.1.3 Governance (ID.GV)

definition

"The policies, procedures, and processes to manage and monitor the organization's regulatory, legal, risk, environmental, and operational requirements are understood and inform the management of cybersecurity risk [29]."

As explained in section A.1.1.2, our solution may be used by any organization, person or company who wishes it. We can not determine this point in advance which is proper to every organization.

A.1.1.4 Risk Assessment (ID.RA)

definition

"The organization understands the cybersecurity risk to organizational operations (including mission, functions, image, or reputation), organizational assets, and individuals [29]."

As explained in section A.1.1.3, our solution can be used by any entity, person or company wishing to use it. Our solution aims to monitor an independent and existing infrastructure by providing the credentials of the targets to monitor.

The organization using our solution must understand the risk of providing every devices credentials to an external monitoring server.

As we cannot answer in advance for all potential users of our solution, we leave it to them to complete this point.

A.1.1.5 Risk Management Strategy (ID.RM)

definition

"The organization's priorities, constraints, risk tolerances, and assumptions are established and used to support operational risk decisions." [29]

The priority of the solution is the ability of monitoring the targets.

As it is a tool which can be used by any organization, we can not further develop its priorities, constraints or risk tolerances.

A.1.1.6 Supply Chain Risk Management (ID.SC)

definition

"The organization's priorities, constraints, risk tolerances, and assumptions are established and used to support risk decisions associated with managing supply chain risk. The organization has established and implemented the processes to identify, assess and manage supply chain risks [29]."

Our proposed solution is the automation of a software deployment and configuration which can be done on a physical device or a virtual one.

Regarding the supply chain risk, we leave this point to be filled by the organization using our solution.

A.1.2 Protect

This function concerns the creation of appropriate safeguard in order to ensure the protection of the organization's assets.

A.1.2.1 Identity Management, Authentication and Access Control (PR.AC)

definition

"Access to physical and logical assets and associated facilities is limited to authorized users, processes, and devices, and is managed consistent with the assessed risk of unauthorized access to authorized activities and transactions." [29]

A.1.2.1.1 PR.AC-1: Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users and processes.

At creation of the virtual machine, a user is created with configured credentials. An additional Zabbix user is needed to run the Zabbix features.

When we face an issue, we destroy and redeploy the whole machine using different password but same usernames.

There are, for now, no mechanism which manages, verifies or revokes existing users and processes at the machine runtime.

We leave the task of handling the identities and credentials up to the organization using our solution in order to allow them to add their own politic.

A.1.2.1.2 PR.AC-2: Physical access to assets is managed and protected.

In our case, we deployed our solution on Azure Microsoft cloud provider. Thus, we transferred the risk to this entity.

However, as any organisation, person or company can use the solution and deploy it on the machine they want (in the cloud like us or on a machine on their premises), we leave it to them to complete this point according to their internal policy.

A.1.2.1.3 PR.AC-3: Remote access is managed.

Following our way to deploy the Zabbix server (on Microsoft azure provider), we only allow the IP address used to deploy the device to access the Zabbix server.

In addition, the server is reachable through SSH, HTTPS (for the GUI) and via Azure provided tool when it is deployed on this platform.

A more in-depth analysis of this point through a pentest for example is required before indicating our compliance with it.

A.1.2.1.4 PR.AC-4: Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties.

The users account have the minimum privileges. The user zabbix can only interact with Zabbix files and the needed database. The user credentials to SSH into the server has the least needed privileges as well.

For now, we do not handle the creation, access configuration and authorization of additionaIl users.

We leave it to the users of our solution to complete this point according to their own policy.

A.1.2.1.5 PR.AC-5: Network integrity is protected (e.g., network segregation, network segmentation).

We only have 1 server which is publicly accessible but set up to only accept connection from the configured IP address (the one used to deploy the machine) and the Microsoft azure account which is used to deploy the server.

Not leaving this machine publicly accessible would be necessary. Thus, in our case, we are not compliant with this point.

A.1.2.1.6 PR.AC-6: Identities are proofed and bound to credentials and asserted in interactions.

In our case, we only have one identitie which is the one who created and accesses the Zabbix server.

A.1.2.1.7 PR.AC-7: Users, devices, and other assets are authenticated commensurate with the risk of the transaction.

The password of the server user, and the Zabbix interface are a 11 long randomly generated characters. The actor deploying the server can modify these parameters if it does not follow the organization policies.

We have not implemented a more complex authentication system because we did not think it was necessary.

A.1.2.2 Awareness and Training (PR.AT)

definition

"The organization's personnel and partners are provided cybersecurity awareness education and are trained to perform their cybersecurity-related duties and responsibilities consistent with related policies, procedures, and agreements." [29]

As we cannot know this in advance for any possible user of our solution, we leave it to that user to fill this section.

A.1.2.3 Data Security (PR.DS)

definition

"Information and records (data) are managed consistent with the organization's risk strategy to protect the confidentiality, integrity, and availability of information." [29]

A.1.2.3.1 PR.DS-1: Data-at-rest is protected.

The whole data retrieved by Zabbix is stored in a Database with the password configured at the machine deployment which is accessible by the Zabbix and Root users.

A further analysis of the data-at-rest should be done before considering ourselves compliant with this point.

A.1.2.3.2 PR.DS-2: Data-in-transit is protected.

For now, only SSH (to log to the server) and HTTPS (to access the Zabbix GUI) protocols are allowed to access the device.

Both of these protocols use encryption.

Finally, the Zabbix server retrieves the monitored elements of the targets by SSH connection as well.

A further analysis of the communication between the devices should be done before considering ourselves compliant with this point.

A.1.2.3.3 PR.DS-3: Assets are formally managed throughout removal, transfers, and disposition.

When the solution is not needed anymore, we delete the whole device with all the data that it contains.

We do not manage any assets.

A.1.2.3.4 PR.DS-4: Adequate capacity to ensure availability is maintained.

The deployment of the server is done on the Microsoft Azure provider.

When the capacity is not good enough, the user can deploy the solution using a stronger virtual machine, but it is not automatically done.

A.1.2.3.5 PR.DS-5:Protections against data leaks are implemented.

No such protection are done at the moment. For time budget reason, we leave this as future work.

A.1.2.3.6 PR.DS-6: Integrity checking mechanisms are used to verify software, firmware, and information integrity.

For time budget reason, we do not have implemented such an integrity checking for the Zabbix software.

Additionally, we trust Microsoft Azure when providing the Debian 11 operational system.

A.1.2.3.7 PR.DS-7: The development and testing environment(s) are separate from the production environment.

We used local machines as a testing environment. The production one is deployed on Microsoft Azure.

A.1.2.3.8 PR.DS-8: Integrity checking mechanisms are used to verify hardware integrity.

We do not have implemented such an integrity checking as we did not have any physical hardware (outside the PC used to deploy the solution).

A.1.2.4 Information Protection Processes and Procedures (PR.IP)

definition

"Security policies (that address purpose, scope, roles, responsibilities, management commitment, and coordination among organizational entities), processes, and procedures are maintained and used to manage protection of information systems and assets." [29]

For now, we used the solution as an external person to a cyber range. We leave this section to the organization using our solution.

A.1.2.4.1 Maintenance (PR.MA)

definition

"Maintenance and repairs of industrial control and information system components are performed consistent with policies and procedures." [29]

A.1.2.4.2 PR.MA-1: Maintenance and repair of organizational assets are performed and logged, with approved and controlled tools.

We do not maintain or repair the solution. When an issue occurs, we delete and redeploy the whole server.

A.1.2.4.3 PR.MA-2: Remote maintenance of organizational assets is approved, logged, and performed in a manner that prevents unauthorized access.

As indicated in the last section, we do not maintain or repair the solution, even remotely. When an issue occurs, we delete and redeploy the whole server.

A.1.2.5 Protective Technology (PR.PT)

definition

"Technical security solutions are managed to ensure the security and resilience of systems and assets, consistent with related policies, procedures, and agreements." [29]

A.1.2.5.1 PR.PT-1: Audit/log records are determined, documented, implemented, and reviewed in accordance with policy.

We do not keep any log. Our solution monitors a machine state to get an alarm when it changes. Once the Zabbix server is destroyed, all the data is lost.

We leave this point to the organization using our solution to allow them to follow their policy.

A.1.2.5.2 PR.PT-2: Removable media is protected and its use restricted according to policy.

We leave this point to be filled by the organization using our solution in order to allow them to implement their own policy. In our case, we did not allow the usage of any removable media.

A.1.2.5.3 PR.PT-3: The principle of least functionality is incorporated by configuring systems to provide only essential capabilities.

We only have one device handling the monitoring of the targets, the creation of the tunnels and the monitoring of our solution itself.

To respect this guideline, we should have 3 virtual devices, one for the tunnels, one to monitor the main server, and the main server.

For time budget reason, and for the sake of practicality for our target audience (instructors and participants), we did no split the solution in 3 separate devices. We thus are not compliant with this point.

A.1.2.5.4 PR.PT-4: Communications and control networks are protected.

We leave this point to be filled by the future user of our solution.

A.1.2.5.5 PR.PT-5: Mechanisms are implemented to achieve resilience requirements in normal and adverse situations.

As explained in section A.1.2.4.2, when there is an issue, we simply destroy and redeploy the whole solution using another IP address.

We do not have implemented features as a load balancing or DDoS protection for example.

A.1.3 Detect

As explained on the NIST website : "The Detect Function defines the appropriate activities to identify the occurrence of a cybersecurity event. The Detect Function enables timely discovery of cybersecurity events." [28]

A.1.3.1 Anomalies and Events (DE.AE)

definition

"Anomalous activity is detected and the potential impact of events is understood [29]."

A.1.3.1.1 DE.AE-1: A baseline of network operations and expected data flows for users and systems is established and managed.

For time budget reason, we did not provide such a data flow. We thus are not compliant with this point.

A.1.3.1.2 DE.AE-2: Detected events are analyzed to understand attack targets and methods.

For now, when anomaly is found, we simply delete and redeploy the solution.

We leave this point to be filled by the user of our solution, in order to indicate they analyze any detected event.

A.1.3.1.3 DE.AE-3: Event data are collected and correlated from multiple sources and sensors.

The most possible data is retrieved by the server itself but it is not done by multiple sources and sensors. We thus are not compliant with this point.

A.1.3.1.4 DE.AE-4: Impact of events is determined.

We did not determine a hierarchy of events. As explained two sections above, when we detect an anomaly, we destroy and redeploy the solution.

A.1.3.1.5 DE.AE-5: Incident alert thresholds are established.

We do not have incident alert threshold. We let the organisation using our solution use their own threshold.

A.1.3.2 Security Continuous Monitoring (DE.CM)

definition

> "The information system and assets are monitored to identify cybersecurity events and verify the effectiveness of protective measures." [29]

We use our monitoring solution to monitor itself. We thus are limited by our implemented modules.

Finally, we follow an IaC paradigm for our solution. This means that as soon as as a security incident is detected, we can destroy and redeploy the whole solution with different credentials.

A.1.3.3 Detection Processes (DE.DP)

definition

> "Detection processes and procedures are maintained and tested to ensure awareness of anomalous events." [29]

We are limited by the modules we implemented to monitor the system itself, this means that we are not compliant with this point.

We leave this section to the entity using our solution.

A.1.4 Respond (RS)

This section is about the respond to a detected event discussed in previous section.

As we are following a IaC paradigm for our monitoring solution, the user simply destroy and redeploy the whole solution after an incident.

We thus are not compliant with this section. We leave it to be filled by the organization using our solution.

A.1.5 Recover (RE)

As explained in section A.1.4, the whole infrastructure is following the IaC paradigm and is destroyed and redeployed when an security event occurs.

If a further recovering process is needed, we leave it to be filled by the organization using our solution in order to adapt it to their own needs.

A.2 Conclusion

As the NIST framework was followed after the development of the solution, we have modified the solution as far as possible (in relation to the time budget).

Indeed, we applied the elements of least privilege, configured the firewall, added encryption of communications (HTTPS), the least amount of programs possible, the listing of necessary processes, the suppression of unnecessary users, and the monitoring of the service itself to ensure its proper functioning.

However, we were not able to follow all the guidelines because of the time budget constraint. The points that do not conform to the framework (and concern us such as ID.AM-3, PR.AC-1, Pr.AC-5, PR.AC-6, PR.PT-3 for example) is left as future work.

Finally, there are several points that we leave blank on purpose (such as ID.AM-4, ID.AM-6, ID.BE, ID.GV, ID.RA,...) which can not be determined at this point and need to be filled by the organization using our solution.

The table A.1 summarises the comparison of the NIST key points.

Table A.1: Summary table of the comparison with the NIST framework

Category unique identifier	Compliance
ID.AM-1	yes
ID.AM-2	yes
ID.AM-3	no
ID.AM-4	none
ID.AM-5	yes
ID.AM-6	none
ID.BE	none
ID.GV	none
ID.RA	none
ID.RM	none
ID.SC	none
PR.AC-1	no
PR.AC-2	none
PR.AC-3	no
PR.AC-4	none
PR.AC-5	no
PR.AC-6	no
PR.AC-7	yes
PR.AT	none
PR.DS-1	no
PR.DS-2	no
PR.DS-3	yes
PR.DS-4	no
PR.DS-5	no
PR.DS-6	no
PR.DS-7	yes
PR.DS-8	none
PR.MA-1	no
PR.MA-2	none
PR.PT-1	none
PR.PT-2	yes
PR.PT-3	no
PR.PT-4	no
PR.PT-5	no
DE.AE-1	no
DE.AE-2	none
DE.AE-3	no
DE.AE-4	no
DE.AE-5	none
DE-CM	no
DE.DP	no
RS	yes
RE	yes

Appendix B

Source code

B.1 Python script for SSH tunnels creation

```
from pytunneling import TunnelNetwork
from time import sleep
import sys
import json
import argparse
import yaml


TUNNEL_PATH = "127.0.0.1" # the tunnel begins on the machine
    executing the script.



def parseArguments():
        """
        User can chose by providing a json file or directly give
            the information by argument.
        Target port of last machine is configurable as well.
        """
        parser = argparse.ArgumentParser()
        # Optional arguments
        parser.add_argument("-f", "--file", help="Json file to
            read.", type=str, default=None)
        parser.add_argument("-j", "--jumps", help="Directly
            provided machines to jump through( ip, user, password
             (ip2, user2, password2))...", type=str, default=None)
        parser.add_argument("-p", "--port", help="SSH port of the
             last machine (by default: 22).", type=int, default
            =22)

        # Parse arguments
        args = parser.parse_args()
        return args


def write_path_to_log(path, local_port):
        """
        Writing the script output to a log file.
        """
```

```
        with open("log.txt", "w") as f:
                line = str("\nTunnel open at " + str(path) + ":"
                   + str(local_port) + "\n")
                f.write(line)


def tunnel_creation(counter, info, port):
        """
        Takes the information (ip of the target, username,
           password) and creates the ssh tunnel.
        """
        while counter < len(info):
                with TunnelNetwork(tunnel_info=info[counter],
                   target_ip="127.0.0.1", target_port=port) as tn
                   :
                        write_path_to_log(TUNNEL_PATH, tn.
                           local_bind_port)
                        print("Tunnel with path : {0} available
                           at 127.0.0.1:{1}".format(TUNNEL_PATH,
                           tn.local_bind_port))
                        counter = counter + 1
                        tunnel_creation(counter, info, port)
                        while True:
                                # Use this tunnel
                                sleep(5)


def extract_data_from_yaml(f = "data.yml"):
        """
        Parse the .yml file containing the information about the
           hosts, the wanted tunnels and the monitoring tasks as
           well.
        """
        with open(f, "r") as file:
                data = yaml.load(file, Loader=yaml.FullLoader)
        list_hosts                              = data["hosts"]
        list_tunnels                    = data["tunnels"]
        list_monitoring_tasks   = data["monitoring"]

        return list_hosts, list_tunnels, list_monitoring_tasks


def create_tunnel_info(hosts, tunnels):
        """
        This functions goes through the "tunnel" and "hosts"
           section of the .yml file to merge them and create the
           tunnels.
```

```
        """
        all_tunnel = []
        counter = 0
        for tunnel in tunnels:
                path = []
                for jump in tunnel["path"]:
                        temp_dic = {}
                        temp_dic["ssh_address_or_host"] = hosts[
                            jump]["ssh_address_or_host"]
                        temp_dic["ssh_username"] = hosts[jump]["
                            ssh_username"]
                        temp_dic["ssh_password"] = hosts[jump]["
                            ssh_password"]
                        path.append(temp_dic)
                # Adding the chosen local machine port on which
                    to bind the tunnel
                path[-1]["local_bind_address"] = (tunnel["
                    local_bind_address"]["address"], tunnel["
                    local_bind_address"]["port"])
                all_tunnel.append(path)
        return all_tunnel




if __name__ == '__main__':

        args = parseArguments()
        for a in args.__dict__:
                # If information is in a .yml file
                if (a == "file") and (args.__dict__[a] == None):
                        hosts, tunnels, tasks =
                            extract_data_from_yaml()
                        tunnel_info = create_tunnel_info(hosts,
                            tunnels)
                        tunnel_creation(0, tunnel_info, 22)
                elif (a == "file") and (args.__dict__[a] != None)
                    :
                        hosts, tunnels, tasks =
                            extract_data_from_yaml(args.__dict__[a
                            ])
                        tunnel_info = create_tunnel_info(hosts,
                            tunnels)
                        tunnel_creation(0, tunnel_info, 22)
```

B.2 .yml Configuration file

```
hosts:
  Machine1:
    ssh_address_or_host: 192.168.0.1
    ssh_username: Machine1Username
    ssh_password: Machine1Password
  Machine2:
    ssh_address_or_host: 192.168.0.2
    ssh_username: Machine2Username
    ssh_password: Machine2Password
  Machine3:
    ssh_address_or_host: 192.168.0.3
    ssh_username: Machine3Username
    ssh_password: Machine3Password
  Machine4:
    ssh_address_or_host: 192.168.0.4
    ssh_username: Machine4Username
    ssh_password: Machine4Password


tunnels:
  - tunnel_name: First_tunnel
    path:
      - Machine1
      - Machine2
    local_bind_address:
      address: "0.0.0.0"
      port: 8889
  - tunnel_name: Second_tunnel
    path:
      - Machine3
      - Machine4
    local_bind_address:
      address: "0.0.0.0"
      port: 8890
  - tunnel_name: third_tunnel
    path:
      - Machine1
      - Machine2
      - Machine4
      - Machine3
    local_bind_address:
      address: "0.0.0.0"
      port: 8891
```

```
monitoring:
  zabbix_server:
    api_address: http:/192.168.0.0/zabbix/
    username: Admin
    password: zabbix

  targets:
    Machine2:
      tunnel: First_tunnel
      command:
        - ls
        - ip a
        - cat /etc/network/interfaces
      files_content:
        - /home/user1/file1

    Machine4:
      tunnel: Second_tunnel
      command:
        - ls
      files_content:
        - /etc/network/interfaces
      open_ports:

    Machine3:
      tunnel: third_tunnel
      command:
        - pwd
        - ls
        - cat /etc/network/interfaces
      files_content:
        - /home/vagrant/file1
        - /home/vagrant/file2

    Machine1:
      port: 22
      files_content:
        - /home/user1/file1
      users:
        - user1
        - user2
      version_info:
      network_interfaces:
```

B.3 Ansible script to deploy the tunnels on remote server

```
---
- hosts: zabbix_server
```

```
  tasks:
    - name: install python3-pip
      apt:
        name: python3-pip
        state: present
        update_cache: yes

    - name: installing python libs
      pip:
        name:
          - sshtunnel
          - pytunneling
          - pyyaml

    - name: copying script
      copy:
        src: ../python_scripts/tunnel_creation.py
        dest: "{{tunnel_script_destination}}"

    - name: copying data
      copy:
        src: ../python_scripts/data.yml
        dest: "{{tunnel_script_destination}}"

    - name: Kill background tunnel process if still running
      shell: "kill $(ps aux | grep 'python3 tunnel_creation.py' |
          awk '{print $2}')"
      ignore_errors: True

    - name: start tunnels with 1 months time before killing the
       process
      shell: "nohup python3 tunnel_creation.py \&"
      async: 2592000
      poll: 0
```

B.3.1 Inventory file

```
---
all:
  hosts:
    zabbix_server:
      ansible_host: 20.224.214.123
      ansible_python_interpreter: /usr/bin/python3
      ansible_user: zabbix_server
      ansible_ssh_pass: MyStrongPassword123
      become: "yes"
      ansible_become_method: sudo
      ansible_become: "yes"
```

```
ansible_become_pass: "MyStrongPassword159"
tunnel_script_destination: "/home/zabbix_server/"
```

B.3.2 Output example

```
funix@pop-os:~/Bureau/final_code/ansible$ ansible-playbook main.yml -i inventory.yml

PLAY [zabbix_server] ***********************************************************************************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************************************************************************
ok: [zabbix_server]

TASK [install python3-pip] *****************************************************************************************************************************************************************************
ok: [zabbix_server]

TASK [installing python libs] **************************************************************************************************************************************************************************
ok: [zabbix_server]

TASK [copying script] **********************************************************************************************************************************************************************************
ok: [zabbix_server]

TASK [copying data] ************************************************************************************************************************************************************************************
changed: [zabbix_server]

TASK [Kill background tunnel process if still running] *************************************************************************************************************************************************
fatal: [zabbix_server]: FAILED! => {"changed": true, "cmd": "kill $(ps aux | grep 'python3 tunnel_creation.py' | awk '{print $2}')", "delta": "0:00:00.021272", "end": "2022-04-07 14:04:00.809498", "msg": "non-zero return code", "rc": -15, "start": "2022-04-07 14:04:00.788226", "stderr": "", "stderr_lines": [], "stdout": "", "stdout_lines": []}
...ignoring

TASK [start tunnels] ***********************************************************************************************************************************************************************************
changed: [zabbix_server]

PLAY RECAP *********************************************************************************************************************************************************************************************
zabbix_server              : ok=7    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=1
```

Figure B.1: Output when executing the ansible playbook

B.4 Python script communicating with zabbix API

```
import yaml
import sys

from pyzabbix import ZabbixAPI, ZabbixAPIException

ZABBIX_SERVICE_PORT = 10500 # Common port used by zabbix.
HOST_NAME = "cyber_range_monitoring_host" # Name of the host
    created for the monitoring tasks.
LINUX_SERVER_GROUP_ID = 2                                    #
    Target to monitor are in the Linux server group
TEMPLATE_ID = 0                                              #
    Monitoring template to use. None in our case.



def extract_data_from_yaml(f = "data.yml"):
        """
        Parse the .yml file containing the information about the
            hosts, the wanted tunnels and the monitoring tasks as
            well.
        """
        with open(f, "r") as file:
                data = yaml.load(file, Loader=yaml.FullLoader)
        list_hosts                              = data["hosts"]
        list_tunnels                    = data["tunnels"]
        list_monitoring_tasks   = data["monitoring"]
```

```
        return list_hosts, list_tunnels, list_monitoring_tasks


def extract_zabbix_server_credentials(tasks):
        """
        Parse the .yml file in order to retrieve Zabbix API
           credentials.
        """
        return(tasks["zabbix_server"])


def get_zabbix_hosts_names(address, username, password):
        """
        Return a list containing all zabbix host names.
        """
        all_host_names = []
        try:
                zapi = ZabbixAPI(address)
                zapi.login(username, password)
                all_hosts = zapi.host.get()

                for host in all_hosts:
                        all_host_names.append(host["host"])
        except ZabbixAPIException as e:
                print(e)
                sys.exit()

        return(all_host_names)


def add_zabbix_host(address, username, password):
        """
        Add host to zabbix server if does not exist yet.
        """
        if HOST_NAME not in (get_zabbix_hosts_names(address,
           username, password)):
                try :
                        zapi = ZabbixAPI(address)
                        zapi.login(username, password)
                        test = zapi.host.get()
                        test = zapi.host.create(
                                host=HOST_NAME,
                                        interfaces={"type": 1,"
                                           main": 1,"useip": 1,"
                                           ip": "127.0.0.1","dns
                                           ": "","port":
                                           ZABBIX_SERVICE_PORT},
```

```
                                        groups={"groupid":
                                            LINUX_SERVER_GROUP_ID
                                            ,},
                                        template={"templateid":
                                            TEMPLATE_ID},
                                        inventory_mode=-1
                                    )
                except ZabbixAPIException as e:
                        print(e)
                        sys.exit()

def get_zabbix_host_id_from_name(zabbix_credentials):
        """
        Return the information list = from the targeted zabbix
            host.
        Returns None if the host does not exist.
        """
        res = None
        zapi = ZabbixAPI(zabbix_credentials["api_address"])
        zapi.login(zabbix_credentials["username"],
            zabbix_credentials["password"])
        hosts = zapi.host.get(filter={"host": HOST_NAME},
            selectInterfaces=["interfaceid"])
        if hosts:
                res = hosts
        return(res)


def get_corresponding_tunnel_port(tunnels, tunnel_name):
        """
        Return the port of the tunnel corresponding to the given
            tunnel name in parameter.
        """
        res = ""
        for tunnel in tunnels:
                if tunnel["tunnel_name"] == tunnel_name:
                        res = tunnel["local_bind_address"]["port
                            "]
        return res



def generate_list_of_targets_with_ip_and_port(hosts,
    tunnel_and_its_port, tasks):
        """
        Takes the list of host, host associated with its tunnels
            and the list of tasks and return a list containing
```

```
            truple [machine_name, machine_ip, machine_port].
        The returned list contain the name of the device and
            where to reach it (with or without a tunnel).

        """
        res = []
        zabbix_server_info = tasks["zabbix_server"]
        for target_name in tasks["targets"]:
                temp = []
                if "tunnel" in tasks["targets"][target_name]:
                        # If the device is reachable through a
                            tunnel
                        temp.append(target_name)
                        temp.append("127.0.0.1")
                        temp.append(tunnel_and_its_port[tasks["
                            targets"][target_name]["tunnel"]]) #
                            Get its correspoding tunnel
                elif "port" in tasks["targets"][target_name]:
                        # If the device does not need a tunnel,
                            we still need its ip and a ssh port.
                        temp.append(target_name)
                        temp.append(hosts[target_name]["
                            ssh_address_or_host"])
                        temp.append(tasks["targets"][target_name
                            ]["port"])
                # Adding the host ssh credentials as well
                temp.append(hosts[target_name]["ssh_username"])
                temp.append(hosts[target_name]["ssh_password"])
                res.append(temp)
        return(res)



def generate_tuple_containing_tunnel_name_and_port(tunnels):
        """
        Takes the list of tunnels and return a dictionnary
            containing the tunnel_name as key and its port as
            corresponding element.
        """
        tunnel_and_its_port = {}
        for tunnel in tunnels:
                tunnel_and_its_port[tunnel["tunnel_name"]] =
                    tunnel["local_bind_address"]["port"]
        return(tunnel_and_its_port)


def create_machine_connection_info(targets_ip_and_port,
```

```
    machine_name):
        """
        Returns  a list [machine_name, ip, port, ssh_username,
            ssh_password] corresponding to given machine_name
        """
        for target in targets_ip_and_port:
                if target[0] == machine_name:
                        # Returns only the target credentials, no
                            monitoring command
                        return target[:5]



def push_zabbix_monitoring_command(zabbix_credentials, task_host,
    info):
        """
        Takes the zabbix server credentials, and all the needed
            info of a single task and push it to the server.
        """
        for command in info[5]:
                task_name = str(info[0] + "_command_" + command )
                key = "ssh.run[{},{},{},'UTF-8']".format(
                    task_name, info[1], info[2])
                try:
                        zapi = ZabbixAPI(zabbix_credentials["
                            api_address"])
                        zapi.login(zabbix_credentials["username
                            "], zabbix_credentials["password"])
                        item = zapi.item.create(
                                hostid= task_host[0]["hostid"],
                                name=task_name,
                                key_=key,
                                interfaceid=task_host[0]["
                                    interfaces"][0]["interfaceid
                                    "],
                                authtype="0",
                                username=info[3],
                                password=info[4],
                                params=command,
                                type=13,
                                value_type=4,
                                delay=30
                                )
                        create_trigger(zabbix_credentials,
                            task_name, key)
                except ZabbixAPIException as e:
                        print(e)
```

```
                    sys.exit()
                print("Added monitoring item with name: " +
                    task_name)



def push_zabbix_monitoring_files_content(zabbix_credentials,
    task_host, info):
        """
        Monitor the md5 hash of given files.
        """
        for file in info[5]:
                task_name = str(info[0] + "_files_content_" +
                    file )
                command = str("md5sum " + str(file))
                key = "ssh.run[{},{},{},'UTF-8']".format(
                    task_name, info[1], info[2])
                try:
                        zapi = ZabbixAPI(zabbix_credentials["
                            api_address"])
                        zapi.login(zabbix_credentials["username
                            "], zabbix_credentials["password"])
                        item = zapi.item.create(
                                hostid= task_host[0]["hostid"],
                                name=task_name,
                                key_=key,
                                interfaceid=task_host[0]["
                                    interfaces"][0]["interfaceid
                                    "],
                                authtype="0",
                                username=info[3],
                                password=info[4],
                                params=command,
                                type=13,
                                value_type=4,
                                delay=30
                                )
                        create_trigger(zabbix_credentials,
                            task_name, key)
                except ZabbixAPIException as e:
                        print(e)
                        sys.exit()
                pprint("Added monitoring item with name: " +
                    task_name)


def push_zabbix_monitoring_users(zabbix_credentials, task_host,
```

```
    info):
        """
        Monitors the existence of given users name.

        """
        for user in info[5]:
                task_name = str(info[0] + "_users_" + user )
                command = str("id -u " + str(user))
                key = "ssh.run[{},{},{},'UTF-8']".format(
                    task_name, info[1], info[2])
                try:
                        zapi = ZabbixAPI(zabbix_credentials["
                            api_address"])
                        zapi.login(zabbix_credentials["username
                            "], zabbix_credentials["password"])
                        item = zapi.item.create(
                                hostid= task_host[0]["hostid"],
                                name=task_name,
                                key_=key,
                                interfaceid=task_host[0]["
                                    interfaces"][0]["interfaceid
                                    "],
                                authtype="0",
                                username=info[3],
                                password=info[4],
                                params=command,
                                type=13,
                                value_type=4,
                                delay=30
                                )
                        create_trigger(zabbix_credentials,
                            task_name, key)
                except ZabbixAPIException as e:
                        print(e)
                        sys.exit()
                print("Added monitoring item with name: " +
                    task_name)

def push_zabbix_monitoring_version_info(zabbix_credentials,
    task_host, info):
        """
        Monitor the general system informations.
        """
        task_name = str(info[0] + "_version_info" )
        command = str("hostnamectl")
        key = "ssh.run[{},{},{},'UTF-8']".format(task_name, info
            [1], info[2])
```

```
        try:
                zapi = ZabbixAPI(zabbix_credentials["api_address
                    "])
                zapi.login(zabbix_credentials["username"],
                    zabbix_credentials["password"])
                item = zapi.item.create(
                        hostid= task_host[0]["hostid"],
                        name=task_name,
                        key_=key,
                        interfaceid=task_host[0]["interfaces
                            "][0]["interfaceid"],
                        authtype="0",
                        username=info[3],
                        password=info[4],
                        params=command,
                        type=13,
                        value_type=4,
                        delay=30
                        )
                create_trigger(zabbix_credentials, task_name, key
                    )
        except ZabbixAPIException as e:
                print(e)
                sys.exit()
        print("Added monitoring item with name: " + task_name)



def push_zabbix_monitoring_open_ports(zabbix_credentials,
    task_host, info):
        """
        Monitor list of open ports.

        """
        task_name = str(info[0] + "_open_ports" )
        command = str("lsof -i -P -n | grep LISTEN")
        key = "ssh.run[{},{},{},'UTF-8']".format(task_name, info
            [1], info[2])
        try:
                zapi = ZabbixAPI(zabbix_credentials["api_address
                    "])
                zapi.login(zabbix_credentials["username"],
                    zabbix_credentials["password"])
                item = zapi.item.create(
                        hostid= task_host[0]["hostid"],
                        name=task_name,
                        key_=key,
```

```
                        interfaceid=task_host[0]["interfaces
                            "][0]["interfaceid"],
                        authtype="0",
                        username=info[3],
                        password=info[4],
                        params=command,
                        type=13,
                        value_type=4,
                        delay=30
                        )
                create_trigger(zabbix_credentials, task_name, key
                    )
        except ZabbixAPIException as e:
                print(e)
                sys.exit()
        print("Added monitoring item with name: " + task_name)


def push_zabbix_monitoring_network_interfaces(zabbix_credentials,
    task_host, info):
        """
        Monitor the network interfaces file.
        """
        task_name = str(info[0] + "_network_interfaces" )
        command = str("cat /etc/network/interfaces")
        key = "ssh.run[{},{},{},'UTF-8']".format(task_name, info
            [1], info[2])
        try:
                zapi = ZabbixAPI(zabbix_credentials["api_address
                    "])
                zapi.login(zabbix_credentials["username"],
                    zabbix_credentials["password"])
                item = zapi.item.create(
                        hostid= task_host[0]["hostid"],
                        name=task_name,
                        key_=key,
                        interfaceid=task_host[0]["interfaces
                            "][0]["interfaceid"],
                        authtype="0",
                        username=info[3],
                        password=info[4],
                        params=command,
                        type=13,
                        value_type=4,
                        delay=30
                        )
                create_trigger(zabbix_credentials, task_name, key
```

```
                )
        except ZabbixAPIException as e:
                print(e)
                sys.exit()
        print("Added monitoring item with name: " + task_name)


def create_trigger(zabbix_credentials, description, key ):
        """
        Creates the trigger for a given monitoring item
        """
        expression = "last(/{}/{},#1)<>first(/{}/{},100000000)
           =1".format(HOST_NAME, key, HOST_NAME, key )
        try:
                zapi = ZabbixAPI(zabbix_credentials["api_address
                   "])
                zapi.login(zabbix_credentials["username"],
                   zabbix_credentials["password"])
                trigger = zapi.trigger.create(
                        description= description,
                        expression=expression,
                        priority = 5
                        )
        except ZabbixAPIException as e:
                print(e)
                sys.exit()
        print("Trigger added on task " + description)



def generate_zabbix_monitoring_tasks(zabbix_credentials,
   zabbix_host_id, targets_ip_and_port, tasks):
        """
        This function take a list contianing all devices name, ip
            to reach it and port, and the monitoring tasks as
           parametre.
        Return a list containing all needed elements to push to
           zabbix api.
        """
        # Always checking if the host is configured, if not, it
           adds it.
        for task in tasks["targets"]:
                for module in tasks["targets"][task]:
                        machine_info =
                           create_machine_connection_info(
                           targets_ip_and_port, task)
```

```
                if module == "command":
                # Basic "command" module chosen
                        machine_info.append(tasks["
                            targets"][task]["command"])
                        push_zabbix_monitoring_command(
                            zabbix_credentials,
                            zabbix_host_id, machine_info)


                elif module == "files_content":
                        # Files_content module chosen
                        machine_info.append(tasks["
                            targets"][task]["files_content
                            "])
                        push_zabbix_monitoring_files_content
                            (zabbix_credentials,
                            zabbix_host_id, machine_info)

                elif module == "users":
                        # Display list of users
                        machine_info.append(tasks["
                            targets"][task]["users"])
                        push_zabbix_monitoring_users(
                            zabbix_credentials,
                            zabbix_host_id, machine_info)

                elif module == "version_info":
                        # Display systeme info
                        machine_info.append(tasks["
                            targets"][task]["version_info
                            "])
                        push_zabbix_monitoring_version_info
                            (zabbix_credentials,
                            zabbix_host_id, machine_info)

                elif module == "open_ports":
                        # Display open ports
                        machine_info.append(tasks["
                            targets"][task]["open_ports"])
                        push_zabbix_monitoring_open_ports
                            (zabbix_credentials,
                            zabbix_host_id, machine_info)

                elif module == "network_interfaces":
                        # Display the network interfaces
                            files
                        machine_info.append(tasks["
```

```
                                        targets"][task]["
                                        network_interfaces"])
                                    push_zabbix_monitoring_network_interfaces
                                        (zabbix_credentials,
                                        zabbix_host_id, machine_info)



if __name__ == '__main__':
        hosts, tunnels, tasks = extract_data_from_yaml()
        zabbix_credentials    = extract_zabbix_server_credentials
            (tasks)

        add_zabbix_host(zabbix_credentials["api_address"],
            zabbix_credentials["username"], zabbix_credentials["
            password"]) # Create the host on which we add the
            tasks

        zabbix_host_id        = get_zabbix_host_id_from_name(
            zabbix_credentials) # retrieve the host id
        tunnel_and_its_port   =
            generate_tuple_containing_tunnel_name_and_port(tunnels
            )
        # Generate the list containing the target, which ip and
            port to join them (depending if we connect to them
            through a tunnel or not)
        targets_ip_and_ports  =
            generate_list_of_targets_with_ip_and_port( hosts,
            tunnel_and_its_port, tasks)


        generate_zabbix_monitoring_tasks(zabbix_credentials,
            zabbix_host_id, targets_ip_and_ports, tasks)
        # Pushing each monitoring task to zabbix server
```

B.4.1 Output example

```
funix@pop-os:~/Bureau/final_code/python_scripts$ python3 zabbix_api_communication.py
Added item with itemid 43270 to host: 10524
Added item with itemid 43271 to host: 10524
Added item with itemid 43272 to host: 10524
Added item with itemid 43273 to host: 10524
Added item with itemid 43274 to host: 10524
Added item with itemid 43275 to host: 10524
Added item with itemid 43276 to host: 10524
Added item with itemid 43277 to host: 10524
Added item with itemid 43278 to host: 10524
Added item with itemid 43279 to host: 10524
Added item with itemid 43280 to host: 10524
Added item with itemid 43281 to host: 10524
Added item with itemid 43282 to host: 10524
Added item with itemid 43283 to host: 10524
Added item with itemid 43284 to host: 10524
Added item with itemid 43285 to host: 10524
Added item with itemid 43286 to host: 10524
funix@pop-os:~/Bureau/final_code/python_scripts$
```

Figure B.2: Output when pushing the task to the Zabbix server.

B.5 Screenshot of Zabbix interface

B.5.1 Page containing all the monitoring tasks.

Figure B.3: Screenshoot showing zabbix interfaces with all the monitoring tasks.

B.5.2 Example of the Alert system

Figure B.4: Screenshot of the User interface with a monitoring alert.

B.5.3 Output of the test on a complex infrastructure

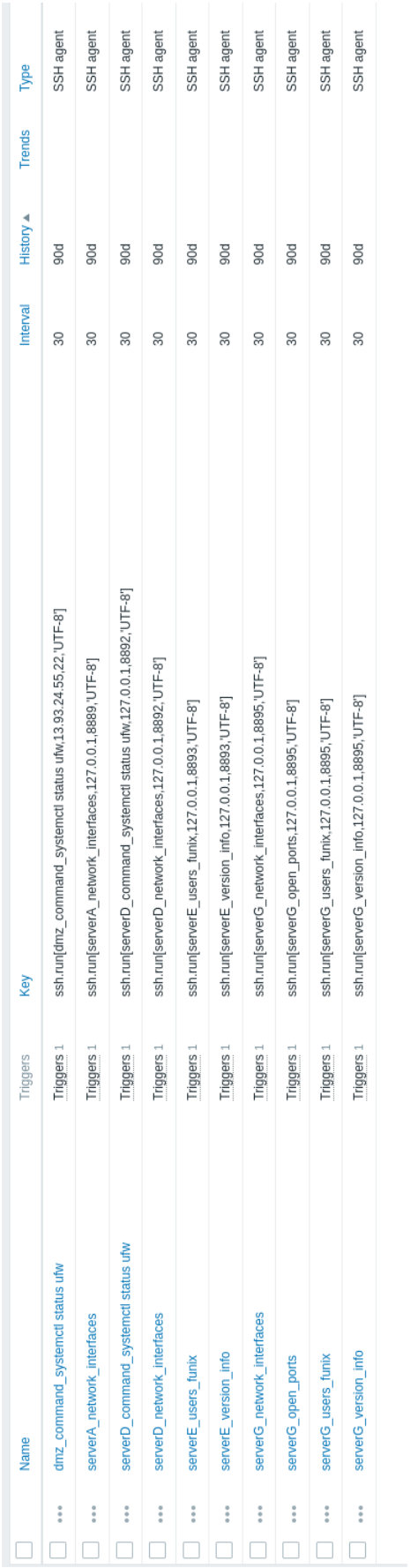

Figure B.5: Screenshot of the generated monitoring task for the complex infrastructure.

Appendix C
Documentation

C.1 Digital format documentation

The whole code and its complete documentation are available on the gitlab https://gitlab.com/Amuranov/automated-monitoring-system-for-cyber-range/-/tree/main/.

Appendix D
Initial project proposal

D.1 Title

Title: Agentless monitoring system for penetration testing oriented cyber range infrastructure.

D.2 Proposal summary

D.2.1 Participants

Researchers(s):

- Muranovic Allan
- Project Director/Owner/Co-Author/Reviewer: Dr. Jérôme Dossogne PhD

D.2.2 Background

The number of security threats is growing each year. For example, according to the **N**ational **C**yber **S**ecurity **C**enter, during the beginning of the pandemic in April 2020, the number of reported incidents in Switzerland was around 350 per week, compared to the usual 100 to 150 [25]. In addition, as Interpol noted in a new study, there is a notable shift in attacker's target from private individuals to large and small businesses, as well as governments and critical infrastructure [18].

These are only 2 reports, but they show how the number of attacks keep growing, and how the attacker focus on bigger target as well.

One of the line of defense is to train cyber security specialist. A popular way to train these specialist is through the usage of cyber ranges.
Using the cyber ranges with variables scenario allows the training of multiple specialist in the wanted subject such as exploiting web services, responding to a live attacks, ... The scenario can be updated as well in order to adapt to new kind of attacks.

In some cases, using a "borrowed" infrastructure (i.e. an infrastructure provided by a specialist as HackTheBox [1] or other cyber ranges creator developed using a proposed scenario), a cyber exercise can be organized by an instructor having the complete write up but without any more privileges on the infrastructure than a simple participant.

Sadly, this impacts the monitoring abilities of the instructor and increases the cost of such monitoring since said instructor would need to develop his own setup using the limited amount of information at his/her disposal. However, knowing the state of certain systems, services, files, etc. helps such instructor ensure that the event progresses smoothly, provide better support, etc.

[1] `https://www.hackthebox.com/`

Additionally, during this kind of security exercises, a pivot is needed to access a given machine or even the whole infrastructure. Indeed, sometimes a vpn configuration is needed or a target machine can be accessible only through a specific one.

These elements make the monitoring even harder, as there are no solution yet to monitor the needed components after jumping from a given device.

Finally, the monitoring must be done without an additional agent on the target. Such agent could implement other vulnerabilities or could give clues to the participant.

Indeed, while network scanners (such as Nmap [2]) allow us to verify to a certain degree that a service is running over the network, as a "participant" (i.e. not a network administrator with direct connectivity, access and all the privilege on all the machine), it becomes more complicated to check that a machine is still in the correct state when it is only available after pivoting from another one. Indeed, it makes the task more tedious as the instructor has to manually pivoting from machine to machine, and, from our experience, typically does not have the time nor will to do so, as he/she needs to focus on the event taking place and thus providing assistance to the participants.
This results in limitation of checking the state of every machine services.

The problem is to find a solution allowing to monitor an infrastructure without having any special rights nor "connectivity" (direct instead of indirect, via pivoting) on it (no more rights needed than the one necessary to complete the scenario).

A solution would be beneficial to the instructor and the participants.

Regarding the instructor, it would be of interest for him to ensure that the "path" to take from machine to machine (i.e. pivoting) is still available and that every needed services to run the scenario are well functioning. And this in real time if possible, making the lost time as small as possible for him and the participants when an issue occurs.

On the participant side, it would allow him to have a growing view of the target infrastructure and to have a better feedback from his manipulation on the system by detecting any impact on it.

D.2.3 Objectives

The goal of this project is to develop a monitoring tool allowing :

- Who: any participant, including an instructor, with limited rights and form of access on an infrastructure (with credentials and/or without, with direct connectivity to various systems or indirect via pivoting) to provide the tool with the information at his disposable (some credentials, routes, etc.) and monitor the state of the systems (provided that the credentials, route etc. allow to do so) without impacting negatively that infrastructure (from a performance and security (C.I.A.-N.R.) point of view).
- the tool could allow such user to share part of the user's view with other users (i.e. an instructor could share information that is collected by the tool using information owned only by the instructor without revealing said information. For instance, the instructor could have knowledge of administrative credentials on a machine X and allow participants to view the list of processes running on said machine without revealing those credentials to the participants) based on certain conditions (e.g. a flag being entered in a CTF platform).

[2] `nmap.org`

By assuming that the instructor has the solution/blueprint to the exercise (i.e. the vulnerabilities to search, the machines users credentials, the exploits and commands to use,...), we want to find a solution allowing him to automatically check for every needed components.

D.2.4 Expected outcome

The project will produce a number of deliverables:

- a master thesis for the Master in Sciences: Cyber Security degree.
- a meta study of existing monitoring solution and, if required, an adaptation/extension/customization/implementation of said tools/platform to suit our requirements (such as automatisation, ease of use, etc. while maintaining full compatibility and upgradability of said underlying tools)
- an instructor/participant monitoring solution for pentest oriented cyber ranges with limited rights.
- a draft of a white paper that can be further modified and improved upon for the research lab to consider publishing at a later date (with or without the student's further collaboration).

D.3 Proposal description (max. 4 pages, ref's included)

D.3.1 Aim of the study and relevance for designated target group

Providing an agentless monitoring tool for pentest oriented cyber ranges. The monitoring tool is useful to instructor and participants with limited access to the infrastructure. In the case where the instructor has the complete solution and all the credentials of the scenario, he is able to check on every needed components of the infrastructure.

At the moment, it is tedious for a cyber range exercise instructor to check for every needed components to complete the scenario.

The monitoring tool can be used by the participants as well. He can check the machine state as well to ensure he did not break a system key point.

D.3.2 State of the art

D.3.2.1 Cyber range monitoring

As founded during a taxonomy review of cyber ranges, there are different methods, dashboard layers and tools used to monitor them. [24].

There are studies focusing every aspect of the monitoring, such as data collection and analysis modules [3] [7] [14], on a dashboard within the infrastructure [2] [9] [14] [21], or even on using layer protocols for data collection [40] [42] [2] [22] [42].

The majority of these monitoring solution use the same tools such as wireshark [37], ADB [37] , Sigar API [9], Nagios [7], Testbed@TWISC Monitor [40].

These are advances in the monitoring system for cyber ranges, but they are all done from the cyber range creator point of view.

At the moment, based on a first exploration of the state of the art, we have not found any conclusive results nor research providing a solution focusing on light touch monitoring for an outsider with a restricted access nor checking for functioning pivoting.

Finally, there are available agentless monitoring tool such as Zabbix, CheckMK or Prometheus, to cite just a few. Once the requirements of the project are defined, these tools will be studied to find out if they are suitable for our problem.

D.3.2.2 Cyber range environment

As defined by Shabeer A., Nicolò M. and Prinetto P. [36], there are 3 different approaches to create a cyber range topology.

A **physical cyber range** which faithfully recreates a network computing infrastructure.

A **virtual cyber range** is when all the components of the reference infrastructure are simulated, using virtualisation technologies, to obtain topologies of different complexity.

A **hybrid cyber range** is sometimes referred as Cyber-Physical Ranges, this typology is a hybrid of the two previous ones.

In our case, we focus on the virtual cyber ranges. In the virtual cyber range category, there are several options for the hypervisor, and/or virtualization techniques.
The infrastructure can be implemented using different kind of supervisor and virtualization

techniques but it does not fundamentally affect the result for the end users (our target group here).

In conclusion, it is good to know the "how" of a cyber range implementation (as we need to try some typical infrastructure), but "what" is inside the cyber range is more important for our research.

D.3.2.3 Pentesting methodology

There are several notorious pentesting framework, such as the OWASP, PTES, NIST,...

As the PTES (The Penetration Testing Execution Standard Documentation) [20] specifies, the actual pentest are usually split in 6 mains points: **(1)** Information gathering, **(2)** threat modeling, **(3)**vulnerability analysis, **(4)** exploitation, **(5)** post exploitation and **(6)** reporting.

In this paper we focus on the information gathering step, vulnerability analysis, exploitation and post exploitation steps to define what is the methodology and the needed tools to complete these steps.

D.3.3 Global research context

This research context is situated in the "automated infrastructure monitoring and testing of cyber range environment". It focus on adding features needed for penetration testing oriented cyber ranges.

D.3.4 Research strategy

The research will be split in 5 main parts :

1. Study main characteristics of potential target infrastructure to take into account. (what typical services are needed on a penetration testing oriented infrastructure, how to automatically check for them with a limited amount of freedom).

2. Instanciation of typical penetration testing infrastructure with an accessible write up when possible (to test the solution on them).

3. Reviewing best tools for a light touch monitoring system.

4. Development of a light touch monitoring system for the instructor, and participant, using the best suited tool.

5. Testing the implementation on typical penetration testing infrastructure (checking for the services and the pivoting when the whole solution is available).

D.3.5 Collaboration

Under the supervision of Mr. Jérôme Dossogne.

D.3.6 Expected outcome

The project will produce a number of deliverables:

- a master thesis for the Master in Sciences: Cyber Security degree.

- a meta study of existing monitoring solution and, if required, an adaptation/extension/customization/implementation of said tools/platform to suit our requirements (such as automatisation, ease of use, etc. while maintaining full compatibility and upgradability of said underlying tools)
- an instructor/participant monitoring solution for pentest oriented cyber ranges with limited rights.
- a draft of a white paper that can be further modified and improved upon for the research lab to consider publishing at a later date (with or without the student's further collaboration).

D.3.7 Feasibility & risks

D.3.7.0.1 Strengths 3 Months Internship in a DevOps position. The researcher has experiences in automation, virtualization (and its networking configuration), provisioning and monitoring of an infrastructure.

The researcher has experiences using Vagrant, Ansible, Proxmox Hypervisor and Terraform. These tool come handy for the instanciation of a typical infrastructure.

D.3.7.0.2 Weaknesses The researcher may have trouble to keep the focus of the research accurate.

The project director can counter this weakness.

D.3.7.0.3 Opportunities The researcher has experiences with, possibly, needed tools.

The researcher has the amount of time available as well.

D.3.7.0.4 Threats Nothing particular to mention.

Overall, the project looks appropriate for the researcher under the supervision of its director.

D.4 Phasing of the project (max. 4 pages)

Start date: 01/01/2022
End date: 30/04/2022

Each of the work packages defined later are subject to change during the course of the research. Nonetheless, they constitute the baseline of the the path to take.

D.4.1 Workpackage 1: Determining needed components to monitor

Start date: 10/01/2022
End date: 20/01/2022

D.4.1.1 Description

To know what to monitor, we need to know the needed components of the infrastructure. These components must be correctly configure to ensure the well functioning of the scenario.

D.4.1.2 S.M.A.R.T. Objectives

- **S**pecific: We need to define all the requirements.
- **M**easurement: We have a list of requirements.
- **A**ssignable: Allan Muranovic
- **R**ealistic: Yes
- **T**ime-related: 2 weeks.

D.4.1.3 Deliverables and their K.P.I.s

To define later

D.4.1.4 Participants and responsibility assignment matrix (RACI model)

- **R**esponsible: **Allan Muranovic**
- **A**ccounted: **Allan Muranovic**
- **C**onsulted: **Jérôme Dossogne**
- **I**nformed:

D.4.2 Workpackage 2: Finding typicals pentest oriented cyber range infrastructure

Start date: 20/01/2022
End date: 30/01/2022

D.4.2.1 Description

By having accesses to typical pentest oriented cyber ranges, the researcher can determine which features must be tested in the target tool. By knowing which features must be tested, the researcher can then find the best way to do it.

In additions, finding typical pentest scenario with a possible complete write-up is welcome as well.
This would allow to have a more efficient testing of the implemented tool later.

D.4.2.2 S.M.A.R.T. Objectives

- **S**pecific: Determining typical target infrastructure.
- **M**easurement: We have a topographie of typical infrastructure.
- **A**ssignable: Allan Muranovic
- **R**ealistic: Yes
- **T**ime-related: 2 weeks.

D.4.2.3 Deliverables and their K.P.I.s

1. We have a typical infrastructure.
2. We have a basic pentest scenario.
3. We have the solution of the pentest scenario.

D.4.2.4 Participants and responsibility assignment matrix (RACI model)

- **R**esponsible: **Allan Muranovic**
- **A**ccounted: **Allan Muranovic**
- **C**onsulted: **Jérôme Dossogne**
- **I**nformed:

D.4.3 Workpackage 3: Instanciation of several typical pentest oriented cyber range infrastructure

Start date: 01/02/2022
End date: 15/02/2022

D.4.3.1 Description

We need standard pentest infrastructure on which to test and implement a solution of light touch monitoring.

D.4.3.2 S.M.A.R.T. Objectives

- **S**pecific: We found and instanciate standard pentest oriented infrastructure
- **M**easurement: Having multiple working pentest oriented infrastructure.
- **A**ssignable: Allan Muranovic
- **R**ealistic: Yes
- **T**ime-related: 2 weeks.

D.4.3.3 Deliverables and their K.P.I.s

1. We have several instantiated infrastructure.
2. Each infrastructure has working features.
3. The features must be standard feature of a pentest target.

D.4.3.4 Participants and responsibility assignment matrix (RACI model)

- **R**esponsible: **Allan Muranovic**
- **A**ccounted: **Allan Muranovic**
- **C**onsulted: **Jérôme Dossogne**
- **I**nformed:

D.4.4 Workpackage 4: Development of tool filling needs defined in point D.4.1

Start date: 15/02/2022
End date: 15/03/2022

D.4.4.1 Description

After defining the needs, we can start building a solution. The solution should allow checking for the infrastructure state, through pivoting if necessary.

D.4.4.2 S.M.A.R.T. Objectives

To define later.

D.4.4.3 Deliverables and their K.P.I.s

To define later.

D.4.4.4 Participants and responsibility assignment matrix (RACI model)

- **R**esponsible: **Allan Muranovic**
- **A**ccounted: **Allan Muranovic**
- **C**onsulted: **Jérôme Dossogne**
- **I**nformed:

D.4.5 Workpackage 5: Reviewing of implementation

Start date: 15/03/2022
End date: 01/04/2022

D.4.5.1 Description

Making a complete review of the implemented tools, its limitation and how it could be improved.

D.4.5.2 S.M.A.R.T. Objectives

- **S**pecific: We want to review the implementation features.
- **M**easurement: We have the limitation, and improvement points of the tool.
- **A**ssignable: Allan Muranovic
- **R**ealistic: Yes
- **T**ime-related: 2 weeks.

D.4.5.3 Deliverables and their K.P.I.s

To define later.

D.4.5.4 Participants and responsibility assignment matrix (RACI model)

- **R**esponsible: **Allan Muranovic**
- **A**ccounted: **Allan Muranovic**
- **C**onsulted: **Jérôme Dossogne**
- **I**nformed:

D.5 Expertise of the project's research team (max. 2 pages)

D.5.1 Expertise

The researcher has participated to CTF. This means he has experiences and knows what tool would be useful or not.

The researcher has experiences with automation regarding servers and virtual machines states, remotely or locally.

The supervisor has experiences in pentesting frameworks and is a teacher at Université Libre de Bruxelles, and Haute École Bruxelles-Brabant.

D.5.2 Publications & porfolio relevant to the project proposal

Gitlab of the automation of server reset at Hetzner server provider: https://gitlab.com/Amuranov/hetzner-automated-server-reset

D.6 Requirement (equipment, skills, ...)

Having references about cyber ranges infrastructure.

The reading of books regarding pentesting framework (such as the NIST pentesint book OWASP...) methdology.

Having these information allow to know which information must be gathered on a pentest oriented cyber range to ensure its proper functioning. The researcher plans to buy them.

D.7 Proposal summary table (max. 2 pages)

Domain: automated infrastructure monitoring and testing of cyber range environment

Study director: Jérôme Dossogne

Research unit/staff/dept: Département d'informatique.

Title: Agentless monitoring system for pentest oriented cyber range infrastructure.

Aim of the study: Development of restricted rights monitoring tool allowing the instructor of a pentest oriented cyber exercise to check for the well functioning of the infrastructure.

Research strategy: Study of State of the art to define actual needs, followed by implementation of the solution finished by reviewing it.

Innovative character: Agentless monitoring of devices after pivoting.

Target group & relevance for that audience: Pertinent for pentester and cyber ranges instructor and participants.

Partnerships: No partnerships.

D.8 Evaluation & Self-Evaluation

D.8.1 Self-Evaluation

- ... /10: Relevance and scope : Is the relevance of the proposed research well defined? Is the scope of project well described and delimited?
- ... /10: Efficiency : Are the required/allocated means means, i.e. personnel, infrastructure and equipment, consistent with the projected outcome of the project?
- ... /10: State of the Art : How is the state of the art, related to this proposal at a national and international level, described? Are the proposers aware of past and current similar research activities?
- ... /10: Research Strategy and Phasing : Are the phases of the project realistic, coherent and in line with the intended objectives? How realistic and well argumented are the objectives? Is the research team well organized and able to perform the planned research
- Remarks:

D.8.2 Evaluation

- ... /10: Relevance and scope
- ... /10: Efficiency
- ... /10: State of the Art
- ... /10: Research Strategy and Phasing
- Remarks: