Deployment of a SOAR open-source tool called the Hive

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Marc Amills I Zamora

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATIONS TECHNOLOGIES AND SERVICES ENGINEERING

Advisors: Oscar Esparza Martin

Marcos Sanchez Martinez

Barcelona, June 2022

Abstract

Nowadays we live in a connected world, where almost every single aspect of our life is connected to the Internet in some way. Although this helps us simplify our daily tasks and become more efficient, it also exposes us to an incredible amount of online threats, and the cost of preventing these threats from becoming an incident can increase exponentially. That is why the concept of SOCaaS is so important.

The aim of this project is to deploy an opensource SOAR solution that can be used by a company to provide an affordable SOC service by sharing resources among several clients. Thanks to the SOAR tools we can centralize the incident management in a single platform and automate it.

In order to demonstrate the capabilities of this system we will also deploy a SIEM tool and a Windows computer to simulate a possible setup that we will attack.

Resum

Avui en dia vivim en una societat connectada, on quasi tots els aspectes de la nostra vida estan connectats a internet d'alguna manera. Encara que això ens ajuda a simplificar les nostres tasques diàries i a ser més eficients, també ens exposa a una gran quantitat d'amenaces digitals i el cost de prevenir que aquestes escalin a un incident pot augmentar de manera exponencial. Per això el concepte de SOCaaS és tant important.

L'objectiu d'aquest projecte és desplegar una eina SOAR de codi lliure que serà utilitzada per una companyia per donar un servei de SOC as a Service a un preu assequible, ja que podrà compartir els recursos entre diversos clients. Gràcies a les eines SOAR, som capaços de centralitzar les amenaces en una sola plataforma i automatitzar-ne la resposta.

Per tal de demostrar les capacitats del Sistema també desplegarem una eina SIEM i un sistema de mostra Windows amb l'objectiu de simular un possible escenari que després atacarem.

Resumen

Hoy en día vivimos en una sociedad conectada, donde casi todos los aspectos de nuestra vida están conectados a internet de alguna forma. Aunque esto nos ayuda a simplificar las tareas diarias y a ser más eficientes, también nos expone a una gran cantidad de amenazas digitales, y el coste de prevenir que estas escalen hasta un incidente puede aumentar de forma exponencial. Por este motivo, el concepto de SOCaaS es tan importante.

El objetivo de este proyecto es desplegar una herramienta SOAR de código abierto que será usada por una compañía para dar Servicio de SOC as a Service a un precio asequible, ya que podrán compartir recursos entre distintos clientes. Gracias a las herramientas SOAR, somos capaces de centralizar las amenazas en una sola plataforma y automatizar su respuesta.

Con el objetivo de demostrar las capacidades del sistema, también desplegaremos una herramienta SIEM y un sistema Windows para simular un posible escenario que luego atacaremos.

Revision history and approval record

Revision	Date	Purpose
0	02/02/2022	Document creation
1	20/05/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	
Marc Amills I Zamora	
Oscar Esparza Martin	
Marcos Sanchez Martinez	

Written by:		Reviewed and approved by:	
Date	02/03/2022	Date	15/06/2022
Name	Marc Amills I Zamora	Name	Oscar Esparza Martin
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

1. Statement of purpose

The purpose of this thesis is to deploy *Thehive*, which is an opensource tool developed by security practitioners, to act as an opensource SOAR to support the EY SOCaaS (Ernest &Young Security Operation Centre as a Service). They have several clients and the SOCaaS will be automated to ingest specific alerts to help the analyst keep track and work on them.

The result of this project is a fully functional *Thehive* in a development environment and a SIEM (Security Information and Event Management) that helps us test its functionalities.

2. Requirements and specifications

For the final product to be production-ready it must achieve the following requirements:

- Should be able to support multi tenancy (multiple organizations).
- Should be connected and configured with a Cortex.
- Must be able to work 24/7.
- Must have an easy deployment.

3. Methods and procedures

This project is using an opensource tool called *Thehive* project created by a group of security practitioners to help smaller companies enter in the cybersecurity world without spending money they might not have in privative software.

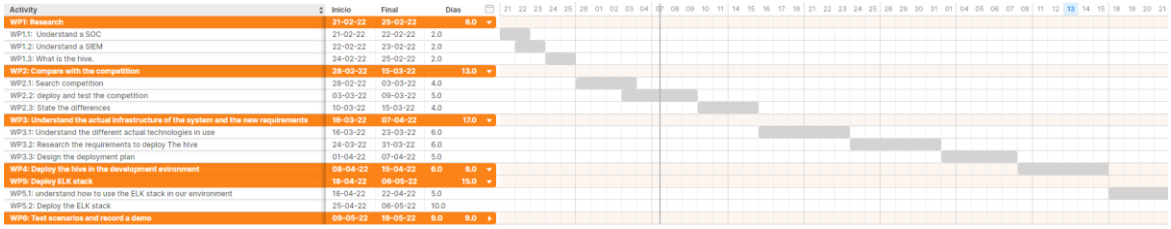
This project uses other opensource tools like, Cassandra by Apache, Lucene engine by Apache, Elasticsearch, Docker, among others, that will be explained in the next sections of this thesis.

The software will be deployed into a rented IaaS (infrastructure as a service) to reduce the costs of this project, although for the testing part will be deployed into virtual machines of my local laptop.

The initial idea was to just deploy the tool but throughout this project new needs emerged, and we deployed other tools to demonstrate and test its capabilities.

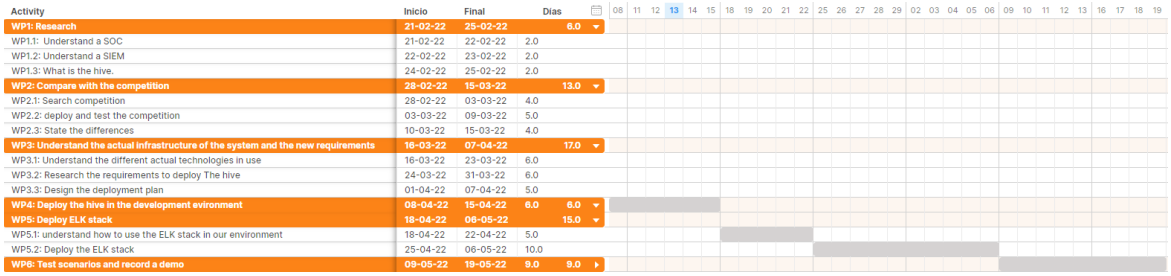
4. Work plan

For this project, a Workplan has been planned to perform all tasks on time. The Workplan contains all the different tasks of each Workpackage and a Gantt Diagram.



Activity	Inicio	Final	Días
WP1: Research	21-02-22	25-02-22	6.0
WP1.1: Understand a SOC	21-02-22	22-02-22	2.0
WP1.2: Understand a SIEM	22-02-22	23-02-22	2.0
WP1.3: What is the hive.	24-02-22	25-02-22	2.0
WP2: Compare with the competition	28-02-22	15-03-22	13.0
WP2.1: Search competition	28-02-22	03-03-22	4.0
WP2.2: deploy and test the competition	03-03-22	09-03-22	5.0
WP2.3: State the differences	10-03-22	15-03-22	4.0
WP3: Understand the actual infrastructure of the system and the new requirements	16-03-22	07-04-22	17.0
WP3.1: Understand the different actual technologies in use	16-03-22	23-03-22	6.0
WP3.2: Research the requirements to deploy The hive	24-03-22	31-03-22	6.0
WP3.3: Design the deployment plan	01-04-22	07-04-22	5.0
WP4: Deploy the hive in the development evironment	08-04-22	15-04-22	6.0
WP5: Deploy ELK stack	18-04-22	06-05-22	15.0
WP5.1: understand how to use the ELK stack in our environment	18-04-22	22-04-22	5.0
WP5.2: Deploy the ELK stack	25-04-22	06-05-22	10.0
WP6: Test scenarios and record a demo	09-05-22	19-05-22	9.0

Figure 1: Gantt chart 1



Activity	Inicio	Final	Días
WP1: Research	21-02-22	25-02-22	6.0
WP1.1: Understand a SOC	21-02-22	22-02-22	2.0
WP1.2: Understand a SIEM	22-02-22	23-02-22	2.0
WP1.3: What is the hive.	24-02-22	25-02-22	2.0
WP2: Compare with the competition	28-02-22	15-03-22	13.0
WP2.1: Search competition	28-02-22	03-03-22	4.0
WP2.2: deploy and test the competition	03-03-22	09-03-22	5.0
WP2.3: State the differences	10-03-22	15-03-22	4.0
WP3: Understand the actual infrastructure of the system and the new requirements	16-03-22	07-04-22	17.0
WP3.1: Understand the different actual technologies in use	16-03-22	23-03-22	6.0
WP3.2: Research the requirements to deploy The hive	24-03-22	31-03-22	6.0
WP3.3: Design the deployment plan	01-04-22	07-04-22	5.0
WP4: Deploy the hive in the development evironment	08-04-22	15-04-22	6.0
WP5: Deploy ELK stack	18-04-22	06-05-22	15.0
WP5.1: understand how to use the ELK stack in our environment	18-04-22	22-04-22	5.0
WP5.2: Deploy the ELK stack	25-04-22	06-05-22	10.0
WP6: Test scenarios and record a demo	09-05-22	19-05-22	9.0

Figure 2: Gantt chart 2

Work packages:

Project: Deployment of The Hive	WP ref: 1	
Major constituent: Research	Sheet 1 of 6	
Short description: Understand how a soc operates, what is a SIEM, what is the hive and how it operates	Planned start date: 21-02-22 Planned end date: 25-02-22	
	Start event: start project End event: WP1.3	
Internal task T1: Understand a SOC Internal task T2: Understand a SIEM Internal task T3: What is the hive.	Deliverables: -	Dates:

Table 1: WP 1

Project: Deployment of The Hive	WP ref: 2
Major constituent: Compare with the competition	Sheet 2 of 6
Short description: Research the commercial solutions and understand the differences	Planned start date: 28-02-22 Planned end date: 15-03-22

	Start event: WP1.3 End event: WP2.3	
Internal task T1: Search competition Internal task T2: Deploy and test the competition Internal task T3: State the differences	Deliverables: 1. List of competition. 2. VMs with competition	Dates: 1. 07/03/22 2. 15/03/22

Table 2: WP 2

Project: Deployment of The Hive	WP ref: 3	
Major constituent: Understand the actual infrastructure of the system and the new requirements	Sheet 3 of 6	
Short description: Understand the actual infrastructure and plan the deployment	Planned start date: 16-03-22 Planned end date: 07-04-22	
	Start event: WP2.3 End event: WP3.3	
Internal task T1: Understand the different actual technologies in use Internal task T2: Research the requirements to deploy The hive Internal task T3: Design the deployment plan	Deliverables: 1.Minimal requirements list 2.Deployment plan	Dates: 1. 14/03/22 2. 07/04/22

Table 3:WP 3

Project: Deployment of The Hive	WP ref: 4
Major constituent: Deploy the Hive	Sheet 4 of 6
Short description: Deploy the hive	Planned start date: 08-04-22 Planned end date: 15-04-22
	Start event: WP3.3

	End event: WP4	
Internal task T1: Deploy the hive in the actual company infrastructure	Deliverables: Working prototype	Dates: 15/04/22

Table 4: WP 4

Project: Deployment of The Hive	WP ref: 5	
Major constituent: Deploy ELK stack	Sheet 5 of 5	
Short description: Deploy the ELK stack in order to integrate it to our soc infrastructure	Planned start date: 18-04-22 Planned end date: 06-05-22	
	Start event: WP4 End event: WP5.2	
Internal task T1: Understand how to use the ELK stack in our environment Internal task T2: Deploy the ELK stack	Deliverables: Working connectors	Dates: 06/05/22

Table 5: WP 5

Project: Deployment of The Hive	WP ref: 6	
Major constituent: Test the system and record a demo	Sheet 6 of 6	
Short description: Test all the system to ensure it functions properly and record a demo video	Planned start date: 09-05-22 Planned end date: 19-05-22	
	Start event: WP5.2 End event: WP6	
Internal task T1: Test different scenarios to trigger an alert and record a video	Deliverables: Tested and production ready tools	Dates: 19/05/22

Table 6: WP 6

5. Incidents and modifications

During the development of this thesis, a new version of the opensource project used was launched, *TheHive* 5. Now it is part of a company called Strangebee and this version is not fully open source, as the “community edition” has limited capabilities that in our case limits its use as you can only have one organization and 5 users.

On the other hand, we will modify a bit the initial project as my deployment is on a development environment, and I do not have access to endpoints where we can apply responders. Instead, we will deploy an opensource SIEM using the ElasticSearch stack that will be explained in the state of the art section.

Another incident emerged when deploying and configuring the *ELK stack* (ElasticSearch stack) as elastic, the company that has created the *ELK stack,* also created several versions of the product one free and several payment variants. To connect the *ELK stack* with *TheHive* we need to use webhooks which is a paid tool inside the stack. In order to solve this issue, we will configure all the ecosystem and in the last step we will start the free trial to use the paid service.

2. State of the art of the technology used:

In this section I will explain the different state of the art technologies that are involved in the deployment of *Thehive,* either in the definitive deployment or other alternatives that should be understood to take the appropriate decision.

The first set of technologies are dedicated to the mechanisms that host applications in our computer systems.

2.1. Virtualization

Virtualization is a way to simulate a computing environment in a more powerful physical machine to tune the needs of our deployment in a more controlled and precise way, therefore, we can be much more efficient and reduce the deployment costs.

This technology has gained importance since the take-off of cloud technologies where the goal is to reduce the cost while being as much efficient as possible.

Virtualization enables these large facilities to accommodate more customers in less space by adjusting the configurations to the costumers needs.

Let us analyse the virtualization architecture:

- Physical host: Actual physical machine that provides the computing power.
- Host OS: The Operating system where we will install our managing software.
- Hypervisor: A hypervisor is the software that runs on our host OS and it is in charge of creating and managing our virtual machines by providing them the assigned resources.
  We can distinguish two kinds of hypervisors:
  - Bare metal, like: Proxmox, Xenserver, Oracle vm server, VMWARE ESXi…, these hypervisors act as light weight operating system installed directly into the host machine.
  - Hosted, like: VirtualBox, VMware workstation, Parallels, QEMU …, these hypervisors are installed on top of an existing OS which consumes more resources.
- Guest OS: The guest operating system.
- Applications: Finally, you would have the applications you want to deploy.

2.2. Containerization

Containerization is the new evolution of virtualization where we only pack the source code and the necessary OS libraries and dependencies required to run our application resulting in a lighter weight deploy. It was created by the developers needs to ensure that their apps could work in a different environment than the one the developer was working without having to redeploy a completely new OS for every app.

The downside of containerization is that it is built for a specific architecture so if you want to run it in a different architecture you need to rebuild it. On the other hand, it

is lightweight and it has a fast deployment on your infrastructure, and more important, it consumes less resources than a full VM with its own OS.

So, its architecture (in Linux) it is different from the virtualization one:

- Physical host: Actual physical machine that provides the computing power.
- Host OS: The operating system where we will install our managing software.
- Containerization engine: It is the one in charge of managing and assigning resources to each one of the different containers. It could be compared to the hypervisor in the virtual machine architecture. The main containerization engine is Docker although it is starting to raise a new competitor called Podman.
- Applications: Lastly, you would have your applications with their required libraries and dependencies.

An important point is that this is the architecture for Docker in Linux because if you deploy it on Windows, it will create a virtual Linux guest host and, therefore, this architecture loses some efficiency.

Another topic to touch briefly is Kubernetes, which is a container orchestrator meant to automate manual processes, act as a load balancer, auto repair containers, and orchestrate storage volumes.

Once we have seen the mechanisms to deploy our applications, I will start to explain the different ones needed to deploy *Thehive*. As you will see, most of the applications we use are designed to be able to run in a distributed environment, to give scalability and redundancy to the system. We will start with the database.

2.3. CASSANDRA

Cassandra is a NoSQL distributed database focused on performance, scalability and speed developed by Apache Software Foundation.

A NoSQL database refers to a non-relational DB which means it is not based on tables like MySQL, PostgreSQL or MariaDB. There are different kinds of NoSQL databases:

- Document databases: Like MongoDB it stores the information on documents like JSON, BSON or XML.
- Graph databases: They store data in a graph structure which is useful if you want to store information of network data like social relationships. One example could be Neo4J.
- Key-value databases: They are the simplest of all the NoSQL databases, each element is stored with an identifier (key) and the information (value). One example could be Riak.
- Column oriented database: In this case, instead of storing information in rows, it stores it in columns. For example, we can find the Cassandra database.

Thanks to its distributed topology, Cassandra can handle a bigger incoming data velocity and it is more reliable, since it does not relay in single point of failure. Due to the distributed capability, it is also easy to scale by adding more nodes to accommodate more clients in our *Thehive* instance.

Now, we will see the different alternatives of data indexing tools that we can use. The main difference is the ability to act as a distributed system.

2.4. LUCENE ENGINE

Although it is used on its own, it is also the base where the Elasticsearch is built from. It is a high-performance text search engine library, written in Java, so it is cross-platform, and it is open source.

It claims that can ingest up to 150 Gigabytes per hour, its minimal RAM requirement is 1 Megabyte. It allows for fast incremental indexing, so it enables the user to add, modify or delete without reindexing the entire database, and lastly the end stored data is 80% smaller compared to the indexed text.

It works by analysing each individual document and generating tokens, which consist of only the relevant content after getting rid of the meaningless words, like is, and, can... Once this is done, the resulting tokens are stored in an inverted index that contains the key (term), the frequency of the term and the document ID where it appears.

By doing this, you can search any word you like without having to do a full text search, as you already know the document where it appears. It also gives you the frequency, which could be useful if you want to do frequency analysis for things like auto completion in search engines.

In our case it is used as an alternative to Elasticsearch to store the data indexes and it is meant to be used in standalone deployments.

2.5. ELASTICSEARCH

Once we have presented Lucene Engine we can see its evolution. Elasticsearch has been built on top of Lucene in order to achieve horizontal scalability. Based on that, we can define it as a distributed opensource search and analytics engine as it uses a structure based on documents instead of tables and schemas and it has an extensive and well documented REST API to store and search data.

It has the same base components as Lucene Engine: documents, indices and inverted indexes. On top of that, we must understand the new added backend components which gives it the distributed functionality:

- **Cluster:** It is the group of one or more nodes that are connected and give the power to the Elasticsearch.
- **Nodes:** It is a single server that is part of a cluster. We have several kinds of nodes:
    - Master node: It controls the Elasticsearch cluster and oversees by adding and removing nodes as well as creating and delating indexes.
    - Data node: They are the nodes responsible of storing data and execute data-related operations like search and aggregation.
    - Client node: It is a sort of load balancer that oversees the cluster by sending the requests to the appropriate node in order to perform the requested operation.

- **Shards:** Elastic enables the user to subdivide the index into multiple sub-indexes fully functional and independent that can be hosted on any node within the cluster. By using shards, we can ensure redundancy, by preventing losses due to hardware failure and increase query capacity by adding new nodes to the cluster.

Although in this project we will use it as an indexing engine, it has many other use cases like website search, infrastructure metrics and container monitoring, security analytics, business analytics, and even as a SIEM, as we will discuss later.

2.6. HDFS with HADOOP

Hadoop is an opensource framework used to efficiently process and store large datasets with a wide range of sizes thanks to its distributed capabilities. It is also known for its low requirements which enables it to run on clusters of commodity hardware.

The importance of Hadoop in the present time is given by its ability to store and quickly process huge amounts of any kind of data. It achieves this by gathering a big amount of computer power from its distributed network. Another thing that it has thanks to its distributed structure is fault tolerance, as it can redirect the jobs if one node gets down, to ensure that the distributed computing does not fail.

Hadoop is not just one application, it is a platform that integrates several components that enables its distributed data storage and processing capabilities.

- Hadoop Distributed File System (HDFS): Which is a distributed file system that is able to run on commodity hardware.
- Yet Another Resource Negotiator (YARN): Which is in charge of managing and monitoring the cluster nodes and its resource usages, as well as deciding what happens on which node. It works by having a central master node that manages the processing requests and then speaks with the managing nodes that execute the tasks.
- MapReduce: It is a framework that helps the programs do the parallel computation on data. It works by taking input data and converting it to datasets that can be computed into key value pairs.
- Hadoop Common: Provides common Java libraries that can be used across all modules.

In our case, it is one of the alternatives presented by *Thehive* project to take care of the file storing. The alternative is a conventional local file system.

Once explained the applications that will be used, it is time to introduce both systems that will compose the project.

2.7. SIEM

SIEM stands for Security Information and Event Management, its main function is to aggregate relevant data from several sources and identify deviations from the norm and take the appropriate actions.

For a SIEM to operate, it needs to be configured to ingest data from several sources such as hosts, endpoint detection and response (EDR), firewalls, routers, switches, VPNs, honeypots...

Once you have configured the ingest sources, you need to configure the alerts from the insights into attacker tactics, procedures and known indicators of compromise. That should be triggered when certain criteria are matched. Although some are already defined, it is a great idea to create some more specific alerts adapted to your infrastructure.

An important part, especially if you plan on combining a SIEM platform with a SOAR to automate tasks, it is to define a clear and concise taxonomy to identify the different alerts. A taxonomy is the process of naming and classifying things, in this case, alerts so it is easier to identify its origin.

An example of a well defined taxonomy can be *company-Tactic-technic-id* to treat the appropriate threats and classify them according to the client so that it can be successfully placed in the SOAR application.

2.8. SOAR

SOAR stands for Security Orchestration Automation and Response, just by the acronym we can begin to understand the main functionalities of these kind of tools. We will analyse them one by one below.

The first part, Orchestration, is due to its ability to organize the SOC by keeping track of the latest alerts and cases, assigning them to each analyst and organization, and finally, it helps to keep track of the different developments from each case.

The second part, Automation, comes from the use of playbooks which establish the procedures to follow by the analysts given a certain TTP (tactics, techniques, and procedures) which helps us identify the threat and act accordingly. Thanks to the widespread use of API in many devices used in the security environment we can automate the tasks in the playbooks, either by using block diagrams in the most advanced SOARs or by coding in others. Therefore, improving the efficiency of our SOC teams by freeing them from repetitive tasks.

Each SOAR tool uses different methods for accomplishing these objectives. In a later chapter of this thesis, we will be comparing the main SOAR tools in order to understand how they implement them.

3. Methodology / project development:

3.1. Look for alternatives

In this chapter we will do an overview of the different tools that could be an alternative to our opensource tool and then we will keep the ones we can get a test/community licence for.

After researching and a superficial dive, we have got access to several of the most important SOAR tools available. In our case we will be deploying and trying Cortex XSOAR by Palo Alto, Siemplify by Google and Phantom by Splunk. These are the most well-known SOAR tools that offer a community edition for cybersecurity practitioners to test and deploy in their infrastructure.

3.2. Deploy all the tools

In order to compare all the alternatives to our opensource tool, we will have to deploy them to have a hands-on experience on them. In order to do that, I have requested a demo licence to all the main companies for their products.

Some of them provided an already setup OVA (Open Virtual Appliance) to be deployed into a virtualization client, while others supplied an installer. This will be a superficial deployment not connected to the actual infrastructure as it is for testing purposes only.

The only one that did not supply an OVA was Cortex that instead, supplied an installer, so we will deploy an Ubuntu server and built our own OVA to test it.

The main goal of this deployment is to see the different functionalities and how are they implemented.

3.3. Compare the tools

- **Splunk phantom**

  Splunk is a company that creates several utilities for your SOC operations. Its main piece of software is their indexer that with some external modules can be used as a SIEM and it is one of the most used in the industry thanks to its high efficiency and versatility. Although it has the same problem that we can see in this SOAR product, it demands an exhaustive training before operating it, as it bases the majority of the interactions on queries encoded in their special language.

  As for the tools it includes, we can see a playbook tab as the other premium SOARs, although I found it harder to use and operate than the one from Palo Alto, but with training and experience you can get great use out of it.

  Like in the other SOARs, we found a large range of apps that deliver great compatibility with the main apps used in the enterprise environment. We also found a lot of out-the-box playbooks that, after some configuring of the connectors, can save you and your SOC team a lot of time in repetitive tasks.

  After speaking with the vendor, I have an estimated price for the software licence that would cost around 50.000 € for a 1 organization 5 users license.

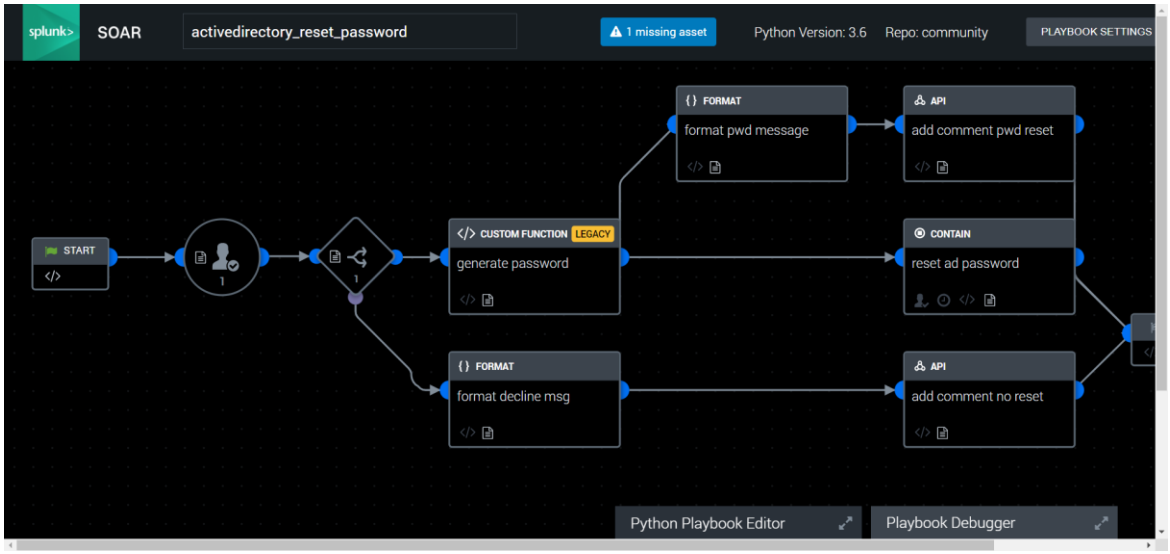


Figure 3: Splunk phantom 1

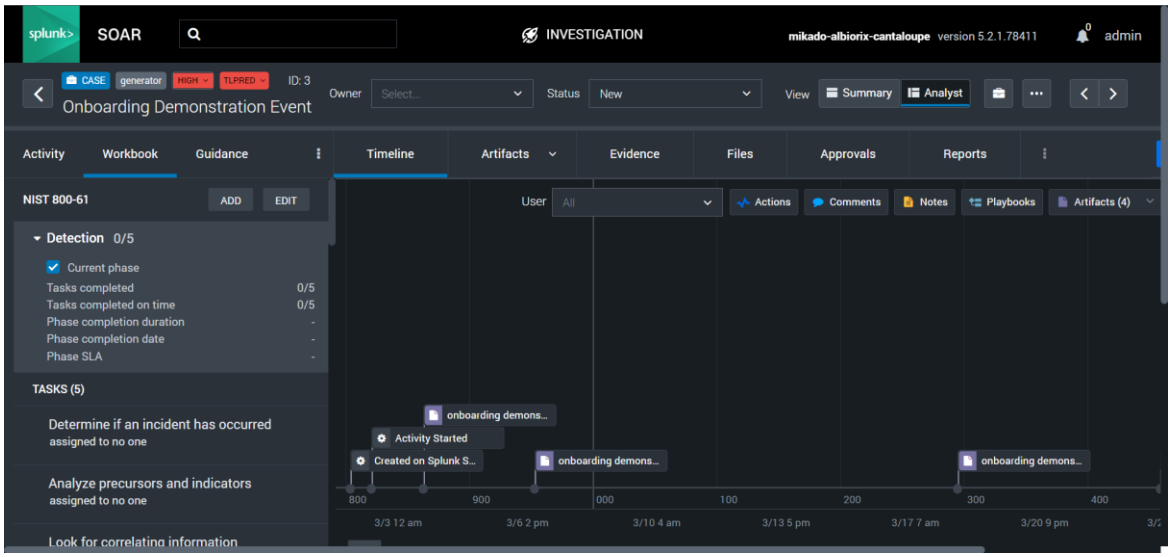


Figure 4: Splunk phantom 2

CONS	PROS
Big learning curve	Very efficient
Not as intuitive as others	Onetime payment
	Integration with other Splunk tools

Table 7: Splunk

- **Palo Alto Cortex xSOAR**

  After reviewing Palo Alto Cortex, we can see that like Siemplify, offers a playbook view that is triggered when an incident matches a certain criterion and contains a set of steps that must be followed in order to mitigate and enrich the knowledge of the incident. The community edition comes with a wide range of playbooks predefined and a lot of integrations for the most well-known security systems manufacturers.

  It also contains a very well detailed *dashboards and reports* screen that helps you understand the actual situation of your organisation. Although I think they tried to put too much information making it overwhelming for the operators.

  Some parts of the software could not be tested as it required the purchase of independent modules such as the threat intelligence to be operational. After some research, we can see that this module would help our teams to have a better understanding of the incident to deliver faster and more accurate response.

  It has an easy-to-use marketplace in the app that allow users to add new integrations and functions developed by Palo Alto, the vendors, community members and yourself, making it capable of interacting with virtually anything you need if you or someone has coded it.

  The *war room* of Cortex can be compared with *investigate* page of splunk where you can see tasks that have been assigned to you and tasks that have already been fulfilled so you can get a global scope about the case.

  Thanks to the page Itprice, we can get an idea of the tool's price. As for all the tools, the price is negotiated with the vendor so they are not publicly available but here, we can see that a perpetual licence would be 312,500$. I was able to speak with a team that is managing an ongoing project to deploy this SOAR tool for a large organization and the cost of the entire project, including tool cost and implementation, adds up to 475.000 €.

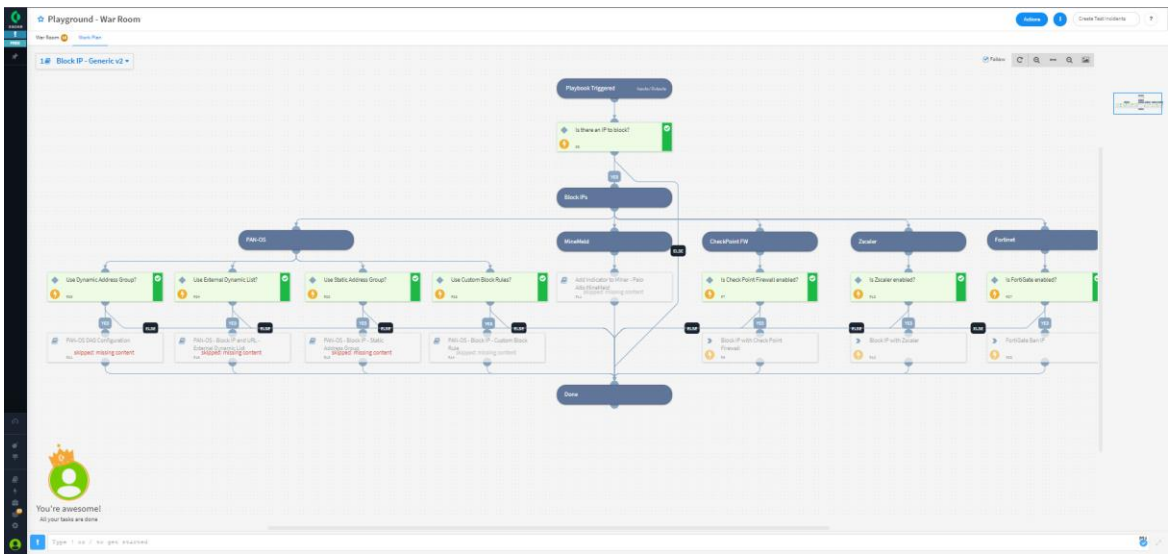


Figure 5: Palo Alto Cortex xSOAR

CONS	PROS
Limited number of users	Great integrations and marketplace
Only one organization included	Wide range of playbooks
Strong user community	Easy to use playbook builder
	Integrated threat intelligence management

Table 8: Palo Alto cortex xSOAR

- **Siemplify**

  Siemplify is a company created in Tel Aviv in 2015 that was acquired by Google this year and has one of the most competent SOAR tools, being able to compete against our other SOAR tools from more mature companies.

  It also features a playbook market and builder using a block building technique, so it adheres to the no code tendency as we have seen in the other SOAR tools.

  According to the AWS marketplace a basic licence for Siemplify giving access to 2 users 7 Playbooks 5 connectors and 100 daily alerts would cost 30.000$ per year. This price is not considering the infrastructure cost, just the licence.

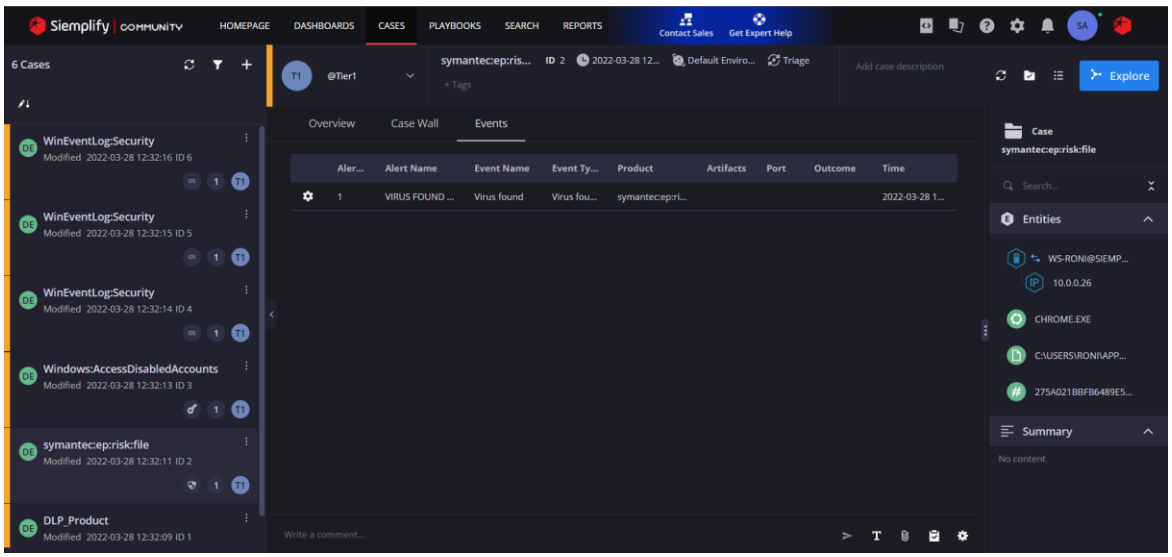


Figure 6: Siemplify 1

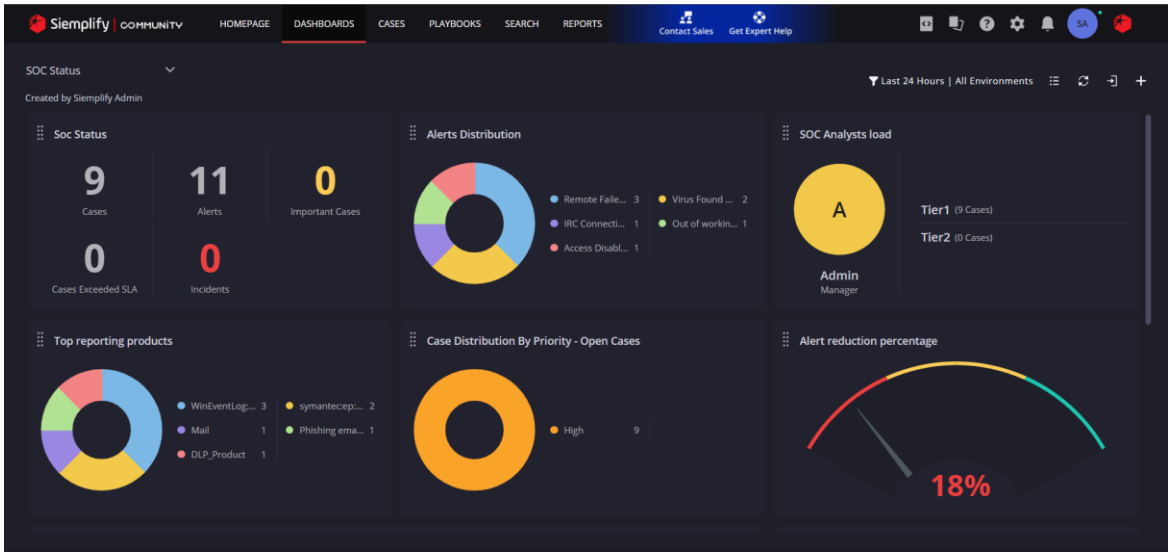


Figure 7: Siemplify 2

CONS	PROS
Younger product	Compatible with the google security products
Smaller community	Better price in cloud deployment with google cloud

Table 9: Siemplify

- **The Hive**

  The last product we will see is *Thehive*, this tool is an opensource tool advertised as a 4 in 1 security incident response platform, although it does not explicitly sell itself as a SOAR, it has all the components to act as it.

  As you will see *Thehive* has two main components, *Thehive* which will act as the orchestrator and Cortex that will handle all the analysis and the automatic response. Because it is an opensource tool, it does not have the budget of the premium tools, and this is present on the final product, as the automation is not as easy and straight forward as with the premium tools.

  On the other hand, being an opensource tool could allow us to modify it and add functionalities on it in a simpler way. Furthermore, thanks to its API you can automate almost any task you want by using simple python scripts and the library they provide.

  Although it is a very complete tool, it is much harder to operate and requires more tinkering by the user for it to act as a SOAR. This is because in this case all the automations are done through the API and it must be coded instead of using simple no-code platforms. It should also be mentioned that there is external software that could help the user create automation in a no-code way such as Shuffle.

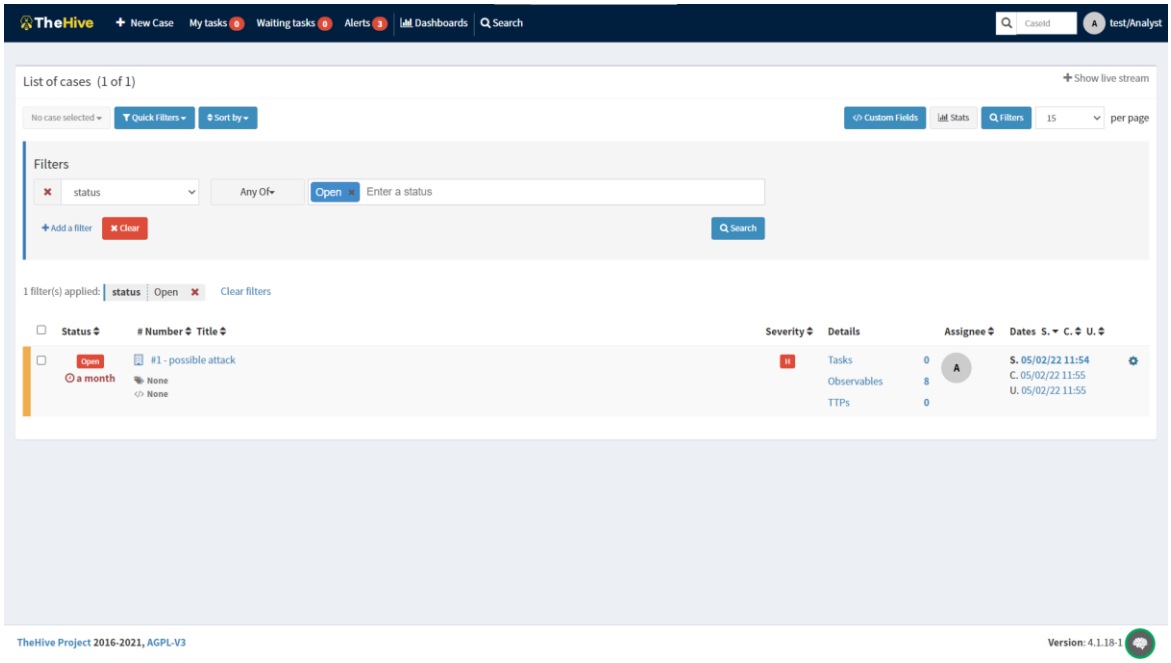


Figure 8: TheHive 1

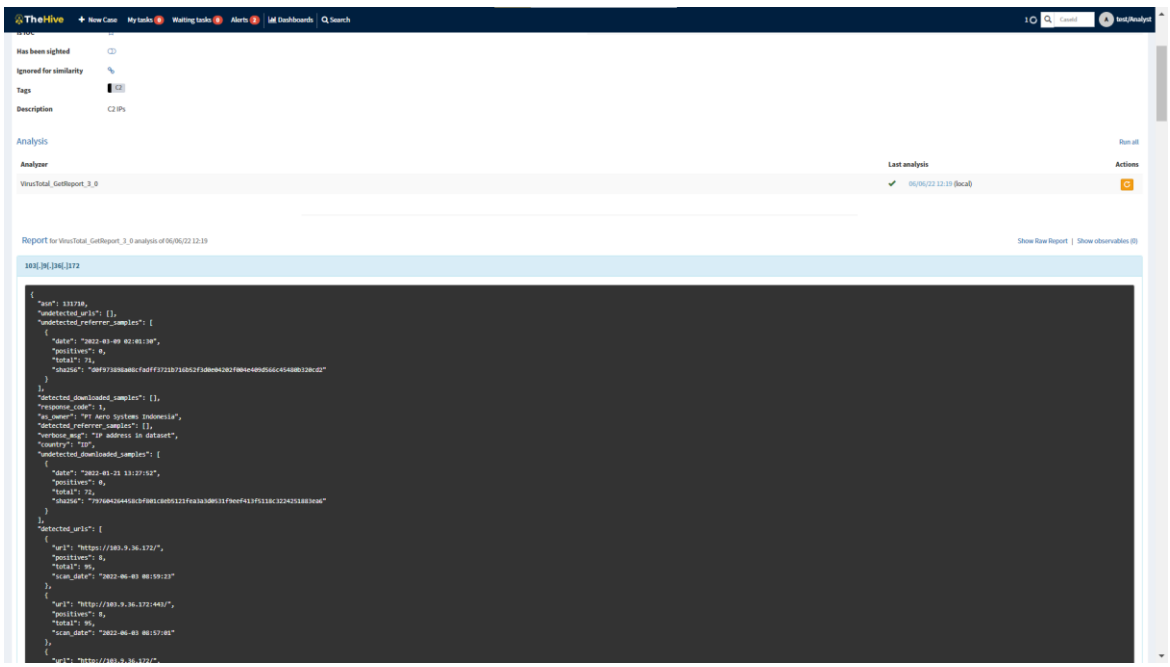


Figure 9: TheHive 2

CONS	PROS
Less intuitive	Free
No support by manufacturer	More customizable
	Supports multitenancy

Table 10: Thehive

Although we can see some differences, all these tools have been built around the same idea: the minimum use of code, the objective of reducing the tasks to be done by analysts while giving them as much information as possible to take a decision on the incident and reducing the response time to minimize the impact on the infrastructure.

3.4. Designing the deployment

In this chapter we will be analysing the connection and interaction between the different tools that we will be deploying.

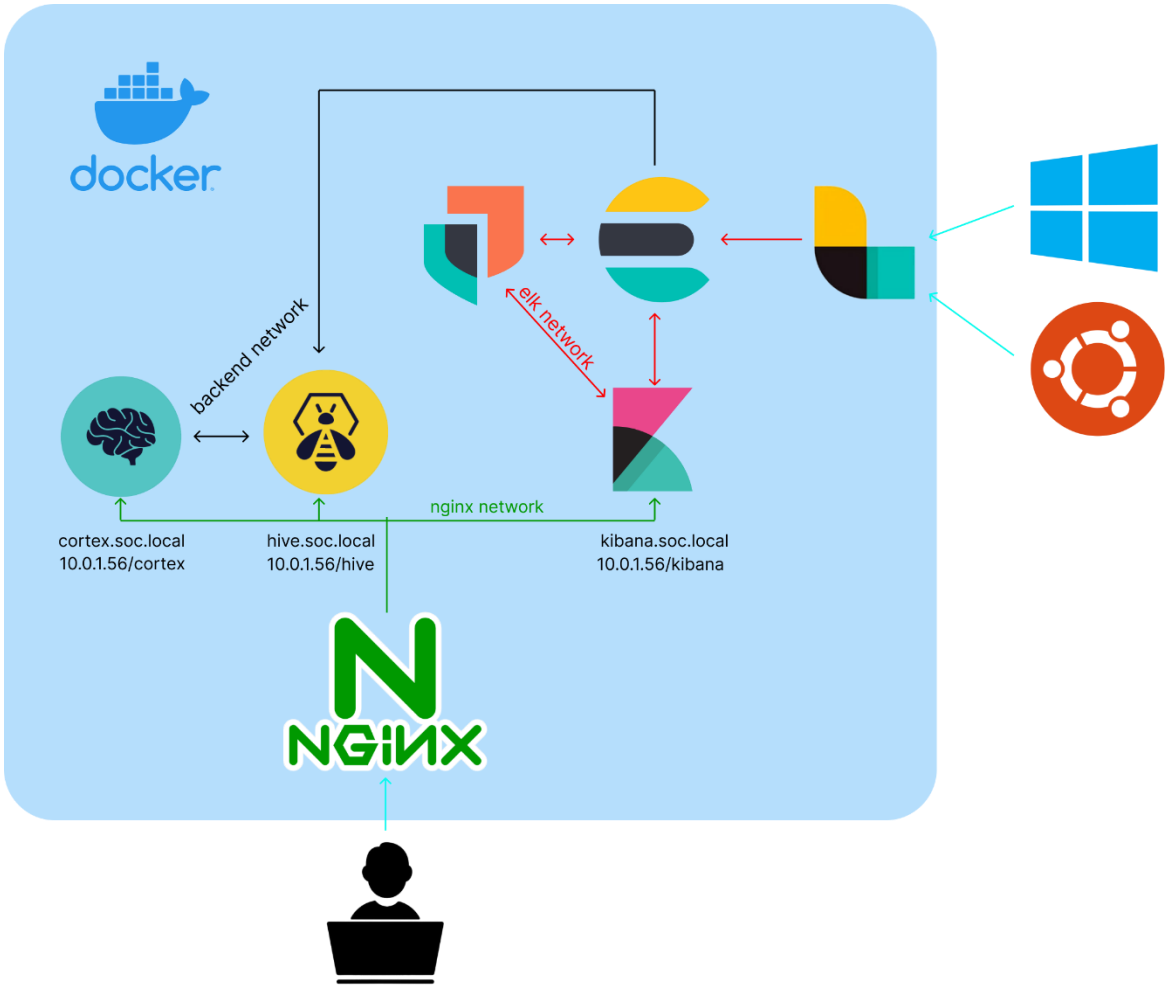


Figure 10: Schema

As I will justify later, for this deployment we will be using Docker, and all the apps will be deployed using docker compose to ensure an easier way to deploy and configure.

We can distinguish 4 main parts in this project:

- The SOAR tool composed by *TheHive* (Orchestrator) and *Cortex* (Responder and Analyser).
- The SIEM tool composed by the ELK stack (Log processor).
- The reverse proxy that is NGINX.

- The test system to send logs (Outside of Docker).

All these parts will be connected using different networks that are grouped by colour in the picture.

- Nginx network (green): It is the network used by nginx to send information to the different frontend tools.
- Backend network (black): It is used by the different services that need to interact between them.
- Elk network is used by the different elements from the ELK stack to interact.
- External network (blue): It is a connection outside Docker used by the user to access the different GUIs and the external systems to send logs.

All the tools will be interacting in a direct or indirect way. On one hand, we have *Thehive* and Cortex, where *Thehive* will act as the orchestration tool and Cortex will be used through *Thehive* to launch the different analysers to investigate the different alerts and cases. Cortex can also be used as a responder to perform actions on the infrastructure to mitigate the threat as fast as possible.

On the other hand, we have the SIEM tools that will be interacting through their own network and is composed by the Elasticsearch that will be in charge of indexing all the parsed logs and perform searches for the user. Then we have Logstash that will be the receiver of the logs from the different systems and will be parsing the logs and forwarding them to Elasticsearch. The last piece is Kibana that will act as our GUI to configure the SIEM and also to enable to user to interact with Elasticsearch to perform the different queries.

The last connection is the interaction between the SIEM and the SOAR. In this case, we will be using a premium feature from Elasticsearch that allows us to interact with a webhook each time an alert is raised. In our case, we will be performing a post request to *Thehive* API to create an alert with all the relevant information of the alert so the analyst can decide if it should be promoted to a case and be investigated.

3.5. Chose the best deployment for the hive

Taking into account our environment and the requirements it has, in this part we will be going through the different technologies that we can use and we will decide which one to use based on our needs and on the knowledge we have acquired on the state-of-the-art chapter.

Let us begin with the deployment environment. We will start with a Linux server image as we do not need a GUI to deploy this project because all the configuration is done via files, which can be perfectly done through the terminal and the end interaction is done through web UIs. If we needed a GUI, we could have used the standard Ubuntu desktop distribution or a Debian distribution.

Once we have the base operating system, we must choose the support for our applications because we can deploy them in a “bare metal way” or in containers. As we will be deploying a lot of different applications and we want to have control over the interaction and an easy deployment, we will be using docker to host our containers for the main applications. If some application does not run well on containers, we will deploy it in “bare metal”. In our case, as it is a development server, it is running inside a VM but in a production environment it would be running

in actual “bare metal”. The alternative of using a VM for each component would consume more resources and would take longer to apply changes.

We will be deploying it by using a tool called docker-compose that enables the user to describe and configure the deployment in a YAML file and then upload it with one single command.

Now we have to determine the best alternatives in *Thehive* deployment configurations, since we have different alternatives for the database and the file storage system.

As a database, we will be using Cassandra inside a docker container in the stack connected through a backbone network from docker without exposing its ports to the rest of the system. We will not be using its distributed capabilities because of the small volume of data, since it would be a waste of resources.

For storage system, we will be using the local system as we are doing a single system deployment and there is no need for distributed storage systems such as Hadoop. The reason is that it will only increase the complexity of the deployment and the system requirements to run the node management mechanisms.

For indexing engine, we will be using a local Lucene engine for *Thehive* and Elasticsearch for Cortex. We will be using Elastic in Cortex because it is the only compatible option. For *Thehive* we will not be using it as it would only increase complexity and workload without added benefit in a standalone system.

3.6. Deployment

In this section we will explain the steps taken to deploy the infrastructure. The files and directories used can be found on the annex zip file along with the demo video.

1. **Preparing the environment:**

   The first step is to deploy and configure the ubuntu server with the appropriate minimum programs such as ssh server, Docker and Docker Compose. Once we have the host system ready, we can start deploying and configuring the different tools.

We will split the deployment in two phases, the SOAR + proxy phase and the SIEM phase.

2. **First phase: SOAR + PROXY**

   - **Launch deploy.sh**

   To start the first phase, we will be using a shell script to automate the deployment, to run the docker compose and other commands that might be necessary to ensure the proper runtime of the system and the different containers.

   - **Launch deploy.sh**

   To automate the process of deployment I created a shell script that takes care of all the pre-deployment steps, the deployment, and the post-deployment configurations.

To do so it will be executing several lines of code that perform the tasks described in this document and in the figure 11.

- **Remove old volume and copy the backup.**

The first step is to remove the vol directory to reset the previous deployment configurations because I founded that redeploying on top of the old volumes caused errors during runtime. Then we copy the configurations from a copy called volback that contains all the necessary configurations and directory structure. For this step, it is important to note that any change that should be permanent should be done in the volback volume.

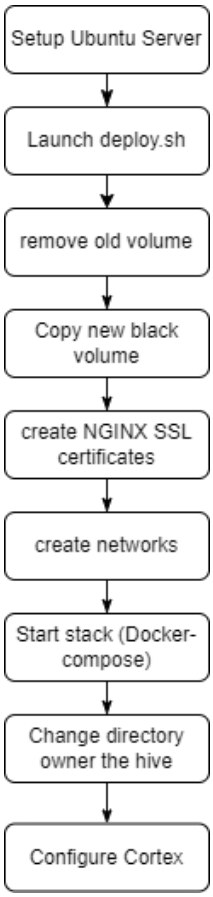


Figure 11: deployment

- **Create the NGINX SSL certificates**

One of the functions of the reverse proxy is to enable the https mode on the services. In order to do so, we have to generate the appropriate certificates that must be placed on the appropriate folder.

If end costumer has SSL certificates that can be applied, this step should be removed and the custom certificates should be placed in the appropriate directory(./volback/ssl/).

- **Create the docker networks**

As I explained on the chapter 3.4, I created 2 backend networks to connect the different pieces of the final deployment, as these networks are defined as external in the docker compose, they need to be created before the containers are deployed.

If the networks are already setup, a message will be displayed indicating it and the networks will remain the same.

- **Start the stack (docker-compose)**

Docker compose is a software developed by Docker that enables us to deploy several containers in a more automatic way and helps us configure them. It also allows us to configure the interaction between them and control the deployment sequence to avoid errors when containers are dependent.

To understand this step, it is important to take a look at the way docker compose works and the basic syntax of this YAML files. We will be taking the phase one docker-file as example.

```
1.  version: '3.8'
2.  services:
3.  #thehive deploy
4.    cassandra:
5.      image: 'cassandra:3.11'
6.      container_name: cassandra
7.      hostname: cassandra
8.      environment:
9.        - MAX_HEAP_SIZE=1G
10.       - HEAP_NEWSIZE=1G
11.       - CASSANDRA_CLUSTER_NAME=thp
```

```
12.     volumes:
13.       - './vol/cassandra/data:/var/lib/cassandra/data'
14.     privileged: true
15.     networks:
16.       - backend
17.
18.   thehive:
19.     image: 'thehiveproject/thehive4:latest'
20.     container_name: thehive4
21.     hostname: thehive4
22.     restart: unless-stopped
23.     depends_on:
24.       - cassandra
25.     ports:
26.       - '9000:9000'
27.     volumes:
28.       - ./vol/thehive/application.conf:/etc/thehive/application.conf
29.       - ./vol/thehive/data:/opt/thp/thehive/data
30.       - ./vol/thehive/index:/opt/thp/thehive/index
31.     privileged: true
32.     command:
33.       --no-config
34.       --no-config-secret
35.       --cortex-port 9001
36.       --cortex-keys ${CORTEX_KEY}
37.       --cortex-proto http
38.       --cortex-hostnames cortex
39.     networks:
40.       - nginxnet
41.       - backend
42.
43. #cortex deply
44.   elasticsearch:
45.     image: 'elasticsearch:7.11.1'
46.     container_name: elkcortex
47.     hostname: elkcortex
48.     environment:
49.       - http.host=0.0.0.0
50.       - discovery.type=single-node
51.       - script.allowed_types=inline
52.       - threat_pool.search.queue_size=100000
53.       - threat_pool.write.queue_size=10000
54.       - ES_JAVA_OPTS=-Xmx2g -Xms2g
55.     networks:
56.       - backend
57.
58.   cortex:
59.     image: 'thehiveproject/cortex:3.1.1-1'
60.     container_name: cortex
61.     hostname: cortex
62.     environment:
63.       - 'job_directory=${job_directory}'
64.     volumes:
65.       - '/var/run/docker.sock:/var/run/docker.sock'
66.       - '${job_directory}:${job_directory}'
67.     depends_on:
68.       - elasticsearch
69.     ports:
70.       - '0.0.0.0:9001:9001'
71.     networks:
72.       - nginxnet
73.       - backend
74. #nginx
75.   proxy:
76.     image: nginx
77.     container_name: "nginx"
78.     hostname: "nginx"
79.     volumes:
```

```
80.       - ./vol/nginx:/etc/nginx/conf.d
81.       - ./vol/ssl:/etc/ssl
82.     ports:
83.       - "80:80"
84.       - "443:443"
85.     networks:
86.       - nginxnet
87.     restart: unless-stopped
88.
89. networks:
90.   backend:
91.     external: true
92.   nginxnet:
93.     external: true
94.
```

To create a working docker compose, first you must state the version of the docker compose syntax, in our case we can see it is 3.8 on row 1.

Then we can start declaring the different services that we will be using. Inside the containers you can set all the necessary configurations, the main ones we will be using are:

- Image: The source of the image for the container.
- Ports: The ports that are going to be exposed, on the left side you place the exposed port, and the right side is the port from the container. This last one will be used to configure the internal docker "router".
- Volumes: This section is used to map folders from the host to the container to achieve persistence and to load configurations that can be changed once the container is already online without having to redeploy.
- Depends_on: It is used to express dependency between services. In our case, we will be using it to link *Thehive* with Cassandra and Cortex with Elasticsearch without exposing its ports to the rest of the system. Ensuring that the deployment order is followed, where the dependent container is not started until the one it depends on is deployed.
- Container_name: Used to state the container name.
- Host_name: Used to state the name of the host inside the docker network.
- Environment: Used to declare environmental variables that will be used inside the container to populate some configuration fields.
- Restart: The restarting policy by default is set so it does not restart if something happens. In our case, we are using unless-stopped to ensure the availability of our system.
- Command: It is used to override the default command.
- Networks: It is used to assign the container to a network.

After the containers we also declare the networks. In our case, as you can see in the last lines, from 89 to 93, we will define two networks in this phase: The backend and the nginxnet which are described as external.

When a network is described as external it is not only in the stack. To use these networks, they must be created manually.

After the first phase of the deployment, we will see that the nginx container is having issues because it will try to communicate with the second phase container called Kibana. This issue will be solved further on the deployment.

- **Change directory owner Thehive**

Once the stack was deployed, we founded out that *Thehive* had an internal problem that was preventing it from having a full deployment. In this case, the two volumes that we had mapped were created by the root user inside the container. For the system to work, they need to be owned by thehive user and this was causing the container to not start properly. That is why we added 2 commands at the end of the previous script to change the ownership of the directories to thehive user.

- **Configure Cortex**

Once all the containers of this phase were deployed and working, we had to log in to cortex and set the admin credentials. After that we had to set up an organization and an account with the org-admin role to get the key for *Thehive* to interact with it.

With that key we must set it in the ./vol/thehive/application.conf file for *Thehive* to be fully integrated with Cortex. Then we restart *Thehive* container that will be forced to reload the configuration.

Once it is running, we logged into the web GUI with the default credentials (admin@thehive.local + secret). On the bottom right we saw the Cortex icon surrounded by a green circle meaning the integration is started and was successful. If the circle was red the integration was started but the connection failed and if the icon is not there the integration is not enabled.

At this point we had a *Thehive* and a Cortex in a functional state. In order for the Cortex to be useful, we had to configure the different analysers and responders according to the infrastructure and the services you have subscribed. This can be used later inside *Thehive*.

3. Second phase: SIEM

Once we have the SOAR tool ready, we will deploy the second phase, a semi open source SIEM tool called Elasticsearch. As stated before, Elasticsearch is an indexer, but thanks to some modules developed by elastic, it can act as a SIEM by indexing all the logs from the different available sources and then applying a set of preconfigured rules to trigger alerts.

In order to deploy this tool, we will use another docker-compose file that can be found inside the docker-elk folder that will allow us to deploy the 3 main elements of the ELK stack. These elements are:

- Elasticsearch indexer that will save all the processed logs.
- Logstash that will receive and parse the logs from our sources.

- Kibana platform that will be our GUI to interact with our SIEM to do queries to our indexer when we need to do some forensic investigation and also to configure the different rules that will trigger the alerts.

In this case, we will be using the command *docker-compose up -d* directly as we will not need to execute any extra command. When all the containers are deployed and they are interacting properly between them, we will have to configure our own alert rules and inside the rule we will configure the webhook to *Thehive*. At this point, we should think carefully about the information that the analyst will be needed to determine if the incident should be promoted to a case and all the IoC that can be used with Cortex.

3.7. Setting up a test environment and testing the deployment

To test and demonstrate the appropriate working of all the pieces of our project, we will have a fresh Windows install where we will be launching a reverse shell with the help of an opensource training tool called Caldera by Mittre. With this tool we can deploy a reverse shell and control the PC with several integrated payloads that should be detected by our SIEM and sent to SOAR for further analysis.

The demo video can be found inside the annex zip in the demo folder and can be used alongside this chapter to help understand the way all the tools interact with each other.

The first step will be to setup the appropriate rules to detect the suspicious activity, in our case we will be using five rules for our demo. On a production environment you would be setting up custom rules and using the already set up by elastic.

- The first rule looks for windows defender logs with the event code = 1116 which identifies that the windows defender has detected malware.
- The second rule looks for windows defender logs with the event code = 5001 which identifies when the protection of windows defender is turned off.
- The third rule looks for an event code = 1102 which identifies when the security logs of the system are cleared.
- The fourth rule looks for an event code = 104 which identifies when the system logs are cleared.
- The fifth rule looks the Microsoft-Windows-PowerShell/Operational log for the event code = 4104 which stores the scripts runed by PowerShell.

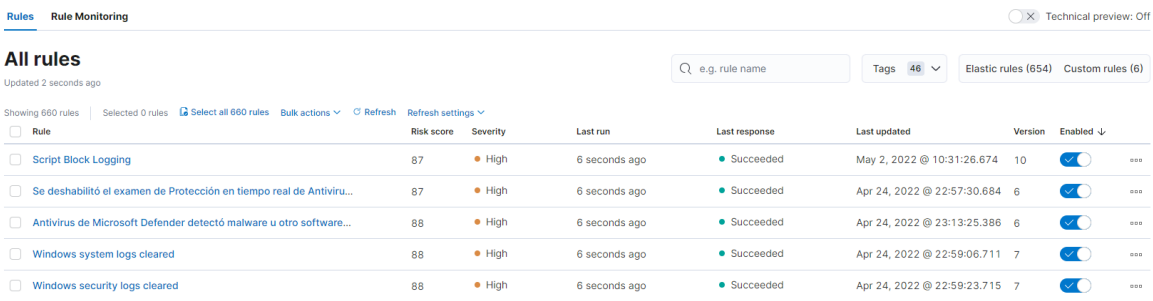


Figure 12: Demo Rules

The second step is to setup a Caldera system that will be our pentesting machine to automate the attack methodologies. Given that this is not the focus of the thesis, we will not be explaining the use of this tool. When this Docker is up and running

we will be trying to deploy the agent on our windows machine. This will trigger the first alert as windows defender will identify the agent as a virus.

The next logical step would be to stop the windows defender which will trigger the next alert on our SIEM and create the appropriate alert on *Thehive*.

Of course, at this point we can install the agent from Caldera. Once the agent is installed, we simulated a predefined attack vector that represents a normal attacker behaviour. In our case, we can see it launches a Mimikatz to retrieve the uses and passwords from the system, as we disabled the antivirus, this software was able to run and retrieve the password. Lastly, we will manually add a logs clearing script that should erase the traces we have left on the system.

Meanwhile the agent installed on windows was sending all the logs to the SIEM tool, thanks to this we can detect the abnormalities as they have triggered several rules. We also can retrieve the logs that the attacker thinks are no longer available and they will be very valuable for the forensic analysts to determine the entry vector and the payload that has deployed.

At the same time, when an alert is triggered it also takes advantage of the SIEM's webhook functionality that enables it to send requests to our SOAR. This simple step will help us achieve the goal of having a single platform of information which will be the SOAR that will contain the alerts. As we can see on the demo, the analyst is able to see all the unprocessed alerts and after analysing the timeframe and the host, he will determine if the threat is real and the next steps that must be followed.

In our case, we cannot execute any responders as we do not have any EDR or firewall connected, but on a production environment, as soon as the analysts sees this anormal activity, they could analyse the hash given by the EDR, enter the SIEM to see the RDP or SSH connection that is executing the commands and take the appropriate actions on the SOAR platform to mitigate the threat. In our case, we could setup a link to an Active Directory to block the user creating the alert as a preventive measure, we could block the IP on the firewall and EDR's among other things.

4. Results

The result of this project can be used in different ways depending on the needs and the tools you want to deploy. The main goal was to set up a SOAR system for a SOCaaS but as a side effect we have prepared a SOC that could be deployed on a small company.

For the first use case, the SOCaaS we would deploy only the SOAR (the first phase) into the company servers and then we should connect the client SIEMS into it in order to get the different alerts. Thanks to the organisation capabilities of *Thehive*, we would create a different organisation for each client so we can compartmentalize the information. To better understand the deployment, you can see the following schema where we can see several clients (red squares) and the SOCaaS server (Blue square). In each client, we would install an agent that would export the alerts from the different SIEMS and perform some automation on the client.

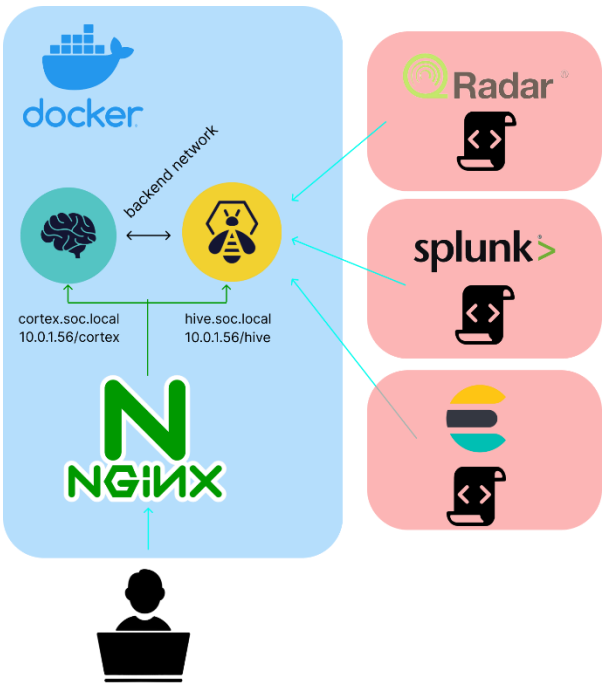


Figure 13: SOCaaS Schema

The second use case, that was not planed initially, is the one that we deployed to test the infrastructure, where we will only deploy one SIEM and connect it to the SOAR. To reduce the cost, we could develop a script to avoid using the premium webhook feature but as this was not the goal of this project this was not done. In the figure 14, you can see an example of configuration where we use a Palo Alto firewall a Windows Active directory for authentication and a Crowdstrike as EDR. These solutions have 2 lines:

- The one coming from Cortex that represents the responses that the analyst can do, such as block a user in the domain, block an IP on the firewall, block an IP on the EDR, ….
- The other lines exiting the solutions are responsible of sending logs to the SIEM for it to process them and find Alerts.

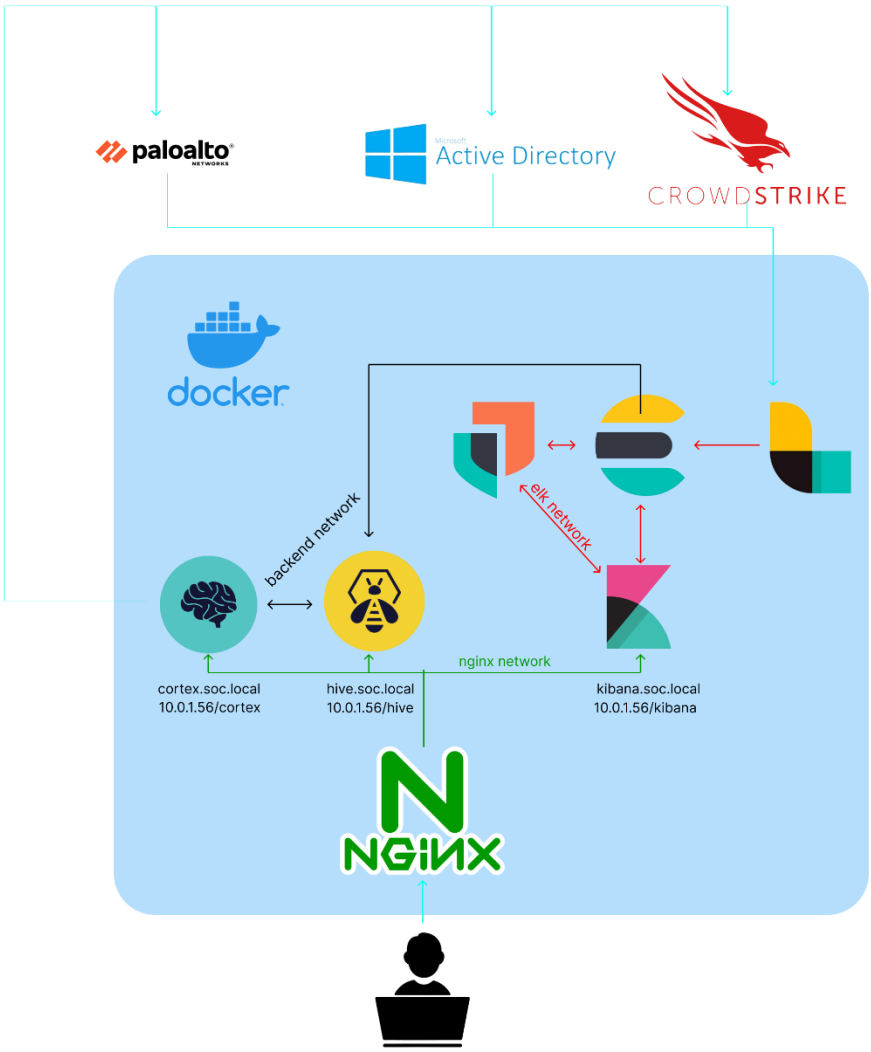


Figure 14: Standalone SOC

The result of this project does not use the Cortex much as it was developed in a development environment, so we did not have any infrastructure to link it to. But in production we would be able to install a Cortex in the client's network and, with the agent script, trigger some automation to respond to certain events in a timeless manner.

5. Budget

Cost of items used:

Item	Value
Server (hetzner AX41)	41.14 €/month → 41.14 x 4 = 164,56 €
Elastic licence	0.5262 €/h → 730,001 h/month → 384,126 €/month

Table 11: Cost items used

Design and prototyping costs:

Task	Hours	Cost
Research	75	675 €
Compare with the competition	70	630 €
Design deployment	105	945 €
Deploy the hive	80	720 €
Deploy ELK	100	900 €
Test	70	630 €
Total		4.500 €

Table 12: Design and prototyping costs

The total approximate cost would be of 4500€ of fixed costs and a monthly maintenance cost of 548,68 €. These are the costs of the prototype. For a real production deployment, the costs should be reconsidered as there would be more deployment and integration costs and the monthly maintenance costs should be higher as the computational need will also increase. The cost of the elastic cloud instance can be estimated through their online calculator: https://cloud.elastic.co/pricing

6. Conclusions and future development:

After all the research and work, I have been able to understand all the parts that are involved on a mature SOC and that are the core of our cybersecurity infrastructure. The first thing I would like to mention is that, in addition to the SOCaaS use case, this deployment can be useful in a small or medium sized company provided that you do not receive a lot of attacks and you have an advanced full-time sys admin that can use the system.

If that is not the case you have two alternatives: go for the more expensive proprietary software, that include more features and are more automated, so they require less technical knowhow. Or you can hire an external contractor to append your infrastructure to their SOC, and they will add the resources, people, and knowledge you need to keep your cybersecurity on shape. On this second alternative is where the first goal of this project gets all its interest.

Even though we could not test the deployment in a production environment, all the requirements were met as it has an easy deployment thanks to the use of docker and docker compose. It supports multi tenancy as in the version 4 it is still open source. It works 24/7 and could be setup in a high availability mode by changing the configuration into a distributed typology. Finally, it has been configured with Cortex without errors.

The next phase of this project would be to deploy this project in the company infrastructure and edit the agents in use to adapt to them to the new environment. It would also require creating our own responders to interact with the client infrastructure over a secure communication channel. All this process would offer a full SOAR tool and we would be able to give the best and most efficient service to our clients.

Bibliography:

[1] PIXEL" El año de los grandes ciberataques en España". *elmundo*, 2021. [Online] Available: https://www.elmundo.es/tecnologia/2021/12/01/61a63b4ae4d4d8db5a8b4577.html. [Accessed: 2 march 2022].

[2] Steve Morgan " Global Ransomware Damage Costs Predicted To Reach $20 Billion (USD) By 2021". *Cybersecurity ventures*, 2019. [Online] Available: https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/. [Accessed: 2 march 2022].

[3] Strangebee, "The hive project". *The hive project*. [Online] Available: https://thehive-project.org/. [Accessed: 22 march 2022].

[4] opensource.com, " What is virtualization?". [Online] Available: https://opensource.com/resources/virtualization/. [Accessed: 22 march 2022].

[5] Ibm "Virtualization". *Virtualization a complete guide*. [Online] Available: https://www.ibm.com/cloud/learn/virtualization-a-complete-guide. [Accessed: 24 march 2022]

[6] Mark Pickavance , Brian Turner " Best virtual machine software of 2022". *Run different operating systems on your PC*. [Online] Available: https://www.techradar.com/best/best-virtual-machine-software. [Accessed: 24 march 2022]

[7] IBM Cloud Education "Containerization". *What is containerization*. [Online] Available: https://www.ibm.com/cloud/learn/containerization. [Accessed: 25 march 2022]

[8] Citrix "https://cassandra.apache.org/_/index.html". [Online] Available: https://www.citrix.com/solutions/app-delivery-and-security/what-is-containerization.html. [Accessed: 25 march 2022]

[9] Apache"Open Source NoSQL Database". [Online] Available https://cassandra.apache.org/_/index.html. [Accessed: 29 march 2022]

[10] Apache" Open Source NoSQL Database". [Online] Available https://cassandra.apache.org/_/index.html. [Accessed: 29 march 2022]

[11] Tutorialspoint "Cassandra". [Online] Available https://www.tutorialspoint.com/cassandra/cassandra_introduction.htm. [Accessed: 29 march 2022]

[12] Datastax " What is Apache Cassandra?". [Online] Available https://www.datastax.com/cassandra. [Accessed: 29 march 2022]

[13] Stephen Watts "Cassandra Introduction: What is Apache Cassandra?". [Online] Available https://www.bmc.com/blogs/apache-cassandra-introduction/. [Accessed: 29 march 2022]

[14] Kelvin "Basic Concepts". [Online] Available https://www.lucenetutorial.com/basic-concepts.html/. [Accessed: 2 april 2022]

[15] orientdb "Lucene FullText Index ".[Online] Available https://orientdb.com/docs/3.0.x/indexing/Full-Text-Index.html/. [Accessed: 2 april 2022]

[16] Stephen Watts "Cassandra Introduction: What is Apache Cassandra?". [Online] Available https://www.bmc.com/blogs/apache-cassandra-introduction/. [Accessed: 2 april 2022]

[17] Ralf Abueg "Elasticsearch: What It Is, How It Works, And What It's Used For". [Online] Available https://www.knowi.com/blog/what-is-elastic-search//. [Accessed: 4 april 2022]

[18] Stephen Watts "Elasticsearch". [Online] Available https://aws.amazon.com/opensearch-service/the-elk-stack/what-is-elasticsearch/. [Accessed: 4 april 2022]

[19] Amazon "What is Hadoop?". [Online] https://aws.amazon.com/emr/details/hadoop/what-is-hadoop/. [Accessed: 10 april 2022]

[20] Sas "Hadoop What it is and why it matters". [Online] https://www.sas.com/en_th/insights/big-data/hadoop.html/. [Accessed: 10 april 2022]

[21] Talend “What is Hadoop?". [Online] https://www.talend.com/resources/what-is-hadoop/. [Accessed: 10 april 2022]

[22] Databricks “Hadoop". [Online] https://databricks.com/glossary/hadoop [Accessed: 10 april 2022]

[23] Shubham Sinha “What is Hadoop? Introduction to Big Data & Hadoop". [Online] https://www.edureka.co/blog/what-is-hadoop/. [Accessed: 4 april 2022]

[24] itprice PALO ALTO PRICE LIST 2022. [Online] https://itprice.com/paloalto-price-list/cortex%20xsoar.html [Accessed: 25 april 2022]

Glossary

[1] SOC: Security Operation Center → Centralized unit tasked to deal with security problems both from a technical and organisational standpoint.
[2] SOCaaS: SOC as a Service → It is a subscription based model for managed SOC operations, that helps companies reduce costs.
[3] SIEM: Security Information and Event Management → It is a tool that supports threat detection, compliance and security incident management by collecting logs from different sources and analysing them.
[4] SOAR: Security Orchestration, Automation, and Response → It refers to a collection of tools that allow organisations to optimize security operations in three main areas: threat and vulnerability management, incident response, and security operations automation.
[5] IaaS: Infrastructure as a Service → Hosting the infrastructure in a hosting provider reducing the deployment and maintenance cost.
[6] EDR: Endpoint Detection and Response → Is an endpoint security solution that continuously monitors end-user devices to detect and respond to cyber threats like ransomware and malware.
[7] RDP: Remote Desktop Protocol → Protocol used to securely communicate with a system by interacting with a remote desktop visualization.
[8] SSH: Secure Shell → Protocol used to securely communicate with a system by interacting with a shell via an encrypted channel.
[9] OVA: Open Virtual Appliance → It is a package that can be used by virtualization application to configure a virtual machine.
[10] API: Application Programming Interface → Allows applications to access data and interact with external software.
[11] TTP: Tactics, Techniques and Procedures → Patterns of activities or methods associated with a specific threat actor or group of threat actors.
[12] OS: Operation System