Master thesis : Honeypot Evolution: Creation Guidelines and Implementation for Third-Party Application Behavior Study Using Cisco SecureX as Monitoring Toolkit

Auteur : Deflandre, Guilian
Promoteur(s) : Donnet, Benoît; 6116
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en informatique, à finalité spécialisée en "computer systems security"
Année académique : 2021-2022
URI/URL : http://hdl.handle.net/2268.2/14580

Honeypot Evolution: Creation Guidelines and Implementation for Third-Party Application Behavior Study Using Cisco SecureX as Monitoring Toolkit

University of Liège

School of Engineering and Computer Science

Academic Supervisor: Prof. Benoît DONNET

Co-Supervisor: Hugues DE PRA

Master's thesis carried out to obtain
the degree of Master of Science in
Computer Science and Engineering

DEFLANDRE Guilian

G.Deflandre@student.uliege.be

Acknowledgements

First, I would like to thank my academic supervisor, Professor Benoît Donnet, for his availability, his advices and his support during this thesis, my university training and even beyond these scopes. I also express my gratitude to my co-supervisor, Hugues de Pra, for his expertise and his trust. I am grateful as well to the entire Cisco team that allowed me to live this exceptional professional experience within this unique company. In particular, many thanks to Senad Aruc who makes this work possible, to Evgeny Mirolyubov and Christopher Van Der Made for their technical advices regarding Cisco's solutions and the rest of the Cisco Amsterdam team for their warm welcome. I want to personally thank Chris Sanders too, who did not hesitate to share his broad experience about honeypots.

Further, I would like to take this opportunity to express my special thanks to my elder brother, without whom I would probably never have got where I am today. His wise advices and constant encouragements have been changing me into a better person every day, for a long time now.

Many thanks also to my girlfriend for her unwavering support and her patience during this process. To all my friends, who have not hesitated to spend time for proofreading or to share their honest opinions regarding my approaches, I also want to convey my deep gratitude. I am aware of being rich in those around me and I will never be grateful enough for their presence in my life.

I am finally very thankful to all the people who directly or indirectly contributed to the completion of this thesis, which is also somehow theirs. To all of them, my sincere thanks.

Abstract

In a world where the cyber threat has never been higher[1, 2], getting to know the adversary is more important than ever. While traditional computer security technologies strive to keep insiders outside the perimeter they defend, honeypots try at all cost to be the primary targets of cyber attacks. They attempt not only to detect these last but also to collect useful information about the black hat community. This thesis aims at defining strong frameworks to create and monitor efficiently the limitless technology that honeypots represent. Through two practical implementations, these frameworks will be used to create two different type of these devices. A first low interaction honeypot will simulate Microsoft's remote desktop protocol for both detection and research. The second is a medium interaction research one feigning an Elastic stack deployment. Relying on the elaborated powerful monitoring framework, efficient strategies will be elaborated using industry IT toolkit to ensure the proper monitoring of these security tools, thus drastically reducing the risk which is too often unfairly associated with them. The data accumulated by these two deployments will show that in a short amount of time, a significant quantity of valuable information, not only for the research community but also for the corporate world, can already be collected by these devices, pointing to their promising future.

Contents

List of Acronyms and Abbreviations

API	Application Programming Interface
ARP	Address Resolution Protocol
ASN	Autonomous System Number
AS	Autonomous System
AV	Antivirus
BYOD	Bring your own Device
CAIDA	Center for Applied Internet Data Analysis
CA	Certificate Authority
CEST	Central European Summer Time
CLI	Command-Line Interface
CSP	Cloud Service Provider
DNS	Domain Name System
DoS	Denial-of-Service
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
HTTP	Hypertext Transfer Protocol
HaaS	Honeypot as a Service
IC3	Internet Crime Complaint Center
IDS	Intrusion Detection System
IP	Internet Protocol
IQR	Interquartile Range
ISP	Internet Service Provider
ITU-T	ITU Telecommunication Standardization Sector
ITU	International Telecommunication Union
IT	Information Technology
IaaS	Infrastructure as a Service
IoT	Internet of Things
L&M	Logging and Monitoring
MFA	Multi-Factor Authentication

NIB	National Internet Backbone
NIST	National Institute of Standards and Technology
OS	Operating System
PDF	Portable Document Format
PNG	Portable Network Graphics
RDP	Remote Desktop Protocol
REST	Representational State Transfer
SME	Small and Medium-Sized Enterprise
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
TPDU	Transport Protocol Data Units
TPKT	Transport Packet
URL	Uniform Resource Locator
VM	Virtual Machine
VPN	Virtual Private Network
XDR	Extended Detection and Response

Introduction

During the afternoon of March 16, 2022, a Webex alert notified an illegitimate connection attempt to a host inside the infrastructure used in this Master's thesis. A foreign service with an IP address located in Russia found the first deployed device described in this writing and initiated a 48-minutes long connection with it, using Microsoft's remote desktop protocol (RDP). This was one of the most exciting news of this work, the security system was working perfectly.

This paradoxical reaction while detecting an insider precisely illustrates the key concept behind honeypots. While most security devices like firewalls, intrusion detection systems (IDS) or antivirus (AV) softwares try to keep threats out, honeypots make every effort to get them in.[3] By simulating legitimate systems, services or tokens, these security devices help to detect (for the production family) or to study (for the research family) behaviors and new trends within the black hat community. In its 2021 cybersecurity research report, Proximus reported that over a sample of 144 Dutch and Belgian companies of all sizes, only 34% of them declared to be absolutely certain that no security incident had occurred in their organization during that year. When crossing the Atlantic ocean, according to the Internet Crime Complaint Center (IC3) annual report, a record number of 847,376 cybercrime complaints addressing Internet scams having affected victims across the globe was filed the same year.[2] In a world where most companies should no longer ask themselves *if* they will be a victim of cybercrime, but rather *when* they will be, detecting and understanding the adversary seems more relevant than ever.

Honeypots are not only useful, but one of their strengths is also their cost effectiveness. In 2021, small and medium-sized enterprises (SMEs) represented 99% of all the European businesses for which budget is crucial. Actually, most of them identified the implementation cost of cyber security solutions as a major challenge.[4] As the reader will be able to discover in this thesis, with some hardware resources, some setups and a bit of creativity, honeypots can be completely configured and help secure an infrastructure at a lower cost.[5, 6]

Everything from the most complex network to the smaller piece of digital data can be transformed into a honeypot, as long as the right creation strategy is used. The variety of these security devices is only limited by the imagination of the one who creates them. Therefore, these systems are completely configurable to meet any kind of needs. This characteristic quality makes honeypots unique in the sense that they represent a very flexible technology used to fulfill different roles and not just a simple tool solving a specific problem.[3]

Part I of this thesis will introduce a theoretical framework guiding the honeypot creation process. In this part, the honeypot taxonomy as well as the strong concepts behind this technology will be defined. Chapter 2 from this part also presents good monitoring practices to adopt when working with honeypots, namely through the definition of a powerful framework. It will also present Cisco SecureX, the monitoring toolkit used in this thesis, and motivate the choice of this application. In today's industry, honeypots are too often associated with a high security risk. Nonetheless, as the reader will be able to discover through this chapter, if honeypots are created and monitored properly, they will not represent a higher risk than any other asset present in the infrastructure where they will be deployed. Indeed, honeypots are exceedingly monitored devices which ensure complete visibility on the events taking place within themselves. One can wonder whether, in the era of the bring your own device (BYOD) advent, the same can be said for each host in an infrastructure.[7, 8]

If the reader is still not convinced of the interest of using honeypots, part II doubtless persuade her or him of it thanks to the two practical deployments it explains. Serving different interests, these more or less complex implementations demonstrate the simplicity and benefits of honeypots for the community and the corporate sector. Through this part, the reader will be able to familiarize herself or himself with the mindset adopted during the creation of honeypots, relying heavily on the theory defined earlier.

As mentioned supra, the main goal of honeypots is to study insiders' behaviors in order to better mitigate threats. The honeypots developed in the practical part II are no exception and were also able to collect metrics about device corruption attempts thanks to both their honey design and the elaborated monitoring strategies. Part III of this writing will present and discuss these collected data and show that, even in a relatively short amount of time, such security machines can already yield a lot of valuable information.

Similarly to every research, this thesis has some limitations. These will be presented in the last part of this thesis, alongside with the general conclusions drawn from the implemented approach adopted in this writing. The methodology was first to define a strong theoretical framework to guide the creation of honeypots and then prove its robustness by using it for real life applications, where information about the black hat community can be extracted. In part IV, the reader will therefore be able to discover if the bet has been held.

State of the Art

The honeypot technology, which first apparition is attributed to Cliff Stoll's book *The cuckoo's egg: Tracking a spy through the maze of computer espionage* released in 1989, is still an active research field nowadays. While this technology was first considered more as a tool for computer security researches, recent contributions show an interest in using this technology as an active security defense.[5, 10–17] In this sense, researches are also oriented nowadays towards the use of honeypots in the Internet of Things (IoT) world to help securing these systems deployed in the industry but also in everyone's homes.[18–20] In fact, some companies have even started to offer honeypot creation services (***a.k.a.*** Honeypot as a Services or HaaS), proposing the design of this type of devices to specially meet the demands of their customers (***e.g.*** application security testing, particular network intrusion detection, etc.).[21, 22]

Even if monitoring is very often mentioned as a key element to ensure usefulness and safety while using honeypots, almost no monitoring framework has been defined ensuring the proper surveillance of such security devices. Actually, relatively recent researches show that implementing efficient system logging and monitoring (L&M) strategies remains a challenge in the industry.[23–26] Some attempts have been made to share good practices related to L&M[3, 5, 26, 27], however most of them are based on concrete examples and do not completely generalize the concepts behind these advices.

The theoretical concepts around honeypots have been stabilized for a few years now. These devices are therefore well-defined and classified, leading to the creation of tools allowing to detect them. One last identified actual trend among recent researches about these devices tackles various anti-honeypot techniques, using well known patterns of these security tools to detect them, thereby breaking the deployed efforts to lure the adversary.[28–31]

By providing guidelines facilitating honeypot creation and by presenting a honeypot deployed to help a company, this Master's thesis proposes a framework to create and use these devices as active security defense mechanisms. Furthermore, it introduces a monitoring framework, defining strong guidelines regarding honeypot logging and monitoring. These guidelines have been kept general on purpose in order to be easily transposed to any other system to be monitored. This writing however does not focus on anti-honeypot contributions, neither on IoT specific study cases for the benefits of presenting an accessible framework helping to build powerful and secure honeypots while not making them complex and completely unrecognizable security devices designed for a specific purpose.

Part I

Theoretical Review

Honeypots Basics

This chapter aims to provide the reader with the knowledge base on honeypots. At first, the honeypots taxonomy is defined, describing the theoretical basis on which this thesis is shouldered. Finally, honeypots' interactivity level, one of the key metrics used while defining honeypots will also be addressed. FIGURE 1.1 summarizes graphically the classification of honeypots defined in the following sections.

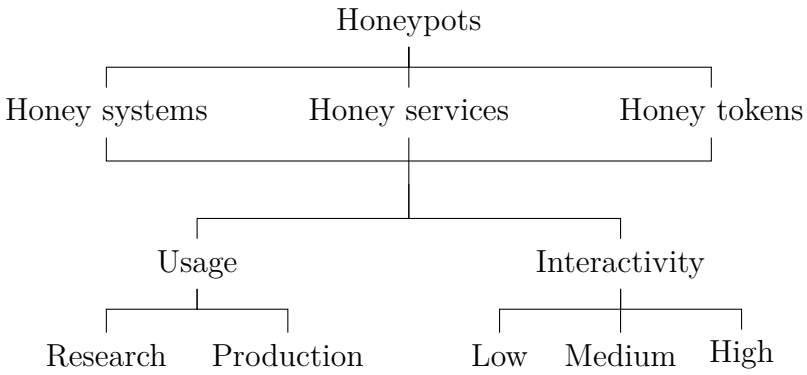


FIGURE 1.1
Classification overview of honeypots introduced in this Master's thesis.[3, 5, 32, 33]

1.1 Definitions

In 2011, R. Joshi and A. Sardana defined honeypots as

> *A program taking the appearance of an attractive service, set of services, an entire operating system or even an entire network, but is in reality a tightly sealed compartment built to lure and contain an attacker.*[32]

L. Spitzner opted in 2002 for a shorter definition

> *A honeypot is a security resource whose value lies in being probed, attacked, or compromised*[3, 34]

Finally, N. Kambow and L. K. Passi, inspired by L. Spitzner, came with a third definition of honeypots in 2014 being

> *Honeypot is a unique security resource which is a part of security mechanism deployed in an organization. These are the resources you want the black hat guys to interact with.*[33]

These definitions, although being very similar, each bring slight differences. In this thesis, each of them will be regarded as valid and one could therefore consider the following general definition: ***A honeypot is a security resource deployed in an organization, taking the appearance of an attractive service or set of services designed to be detected, attacked or compromised in order to detect and analyze security threats.***

From this definition, one can directly extract the mains characteristics of honeypots. They are security resource within an organization, meaning that they are part of a security plan aiming to reduce the security risk. Honeypots should be attractive, a person having access to it should therefore find interesting material inside this service ***e.g.***, an application easy to compromise, files appearing as sensitive (see Chapter 3), etc. Honeypots should also be detectable, their placement inside the organization can therefore be a key factor of their effectiveness. Finally, these devices are made to be attacked or compromised which reflects the fact that the expected interactions with these resources should be perpetrated by attackers.

1.2 Classification

From the definition made in section 1.1, one can understand the fact that the term *resource* is broad which is completely intentional since honeypots can come in many forms. In *Intrusion Detection Honeypots: Detection Through Deception*[5], C. Sanders defines three primary types of honeypots which are the honey systems, services and tokens.[36, 37]

- A *honey system* mimics the interaction of an operating system (OS) and all the services it provides. These are complete computing environments offering all functionalities embedded with the OS they run (Windows 10, Ubuntu 20.04, Debian 11, etc.).
- A *honey service* impersonates the interaction of a specific software or protocol. Of course, several honey services can be used to simulate an entire system. These include basic hypertext transfer protocol (HTTP) honeypots like bap - HTTP Basic Authentication honeypot[38], specially configured Windows Remote Desktop application, etc.
- A *honey token* imitates a legitimate piece of data. This type of honeypot is explored in more details in Chapter 3.

Now that the term *resource* has been refined, it is time to define the two main families, more widely adopted in the literature. These are two in numbers being production and research honeypot families.[3, 5, 32, 33]

1.2.1 Research Honeypots

In the scientific community, when a novelty is discovered, its characteristics are often studied using controlled tests. As an example, consider the discovery of a new type of

virus.[5] A virus will be studied on cell samples in a very controlled way. During all these experiments, researchers will be able to analyze how the virus is spreading, how quickly, on which cell part it will have the most impact, etc. Based on these studies, scientists make assumptions about the correct treatment for this virus.

Research honeypots aim to do the same for the computer security research community. They represent controlled digital test environments allowing to study new types of cyber attacks. As for all experiments, these environments of course try to mimic real life conditions of execution in order to draw the most useful conclusions possible. These honeypots are used to better understand a cyber threat and most importantly, to define how to improve defenses against it. Traditionally, research honeypots are deployed by universities, security research companies, military or government agencies in order to help secure resources indirectly.[3, 33]

Since the goal of these honeypots is to study the black hat community, they need to be easily accessible. FIGURE 1.2.1 shows a typical research honeypot deployment where the honeypot is placed before all internal security devices of a network and directly connected to the Internet.

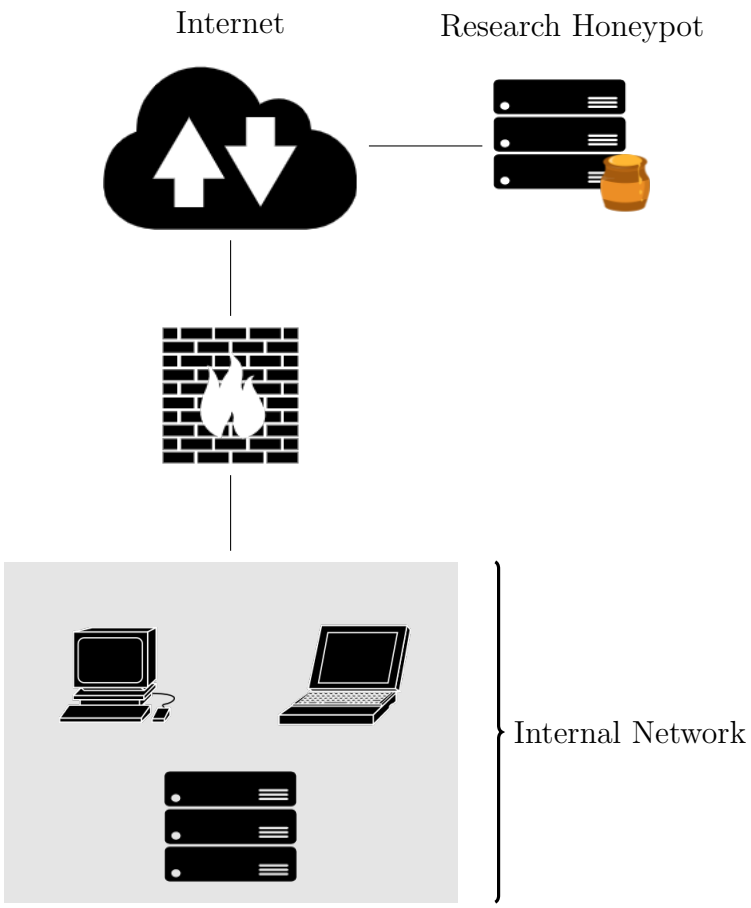


FIGURE 1.2.1
Typical deployment for research honeypot, adapted from [5]. The honeypot is directly connected to the Internet and therefore placed outside any internal network.

1.2.2 Production Honeypots

To prevent threatening situations, nowadays society is equipped with a treasure trove of detection systems. Think about the car industry for instance. When one buys a new vehicle, it is equipped with an alarm system which will activate in the event of an attempted theft. For the newest cars, an anti carjacking system has even been added as a security measure. All of these tools help make the user feel safer and help catch burglars even if their existence is well known.

Production honeypots (***a.k.a.*** detection honeypots) also meet this security need for organizations deploying them. These devices help detect malicious behavior inside the internal network of an organization (***cf.*** FIGURE 1.2.2). These honeypots appear as legitimate parts of an infrastructure, but will be highly monitored in order to detect any interaction with them. Indeed, if such a machine is deployed inside a network, its only purpose should be to detect an attempted interaction from an insider. The easiest way to avoid false positive detection case (think about a car alarm triggered by the passage of a snow plow) is therefore to not use this device for other purposes. Consequently, any activity recorded in this machine will undoubtedly be the one of an unauthorized user. This will be discussed in more details in Chapter 2.

Compared to research honeypots, production honeypots are usually easier to deploy since they require less functionalities. They also generally represent a lower risk even though being generally placed in the internal network of an organization. This is namely due to the fact that they are more difficult to use to harm other systems (again thanks to the limited features).[3]

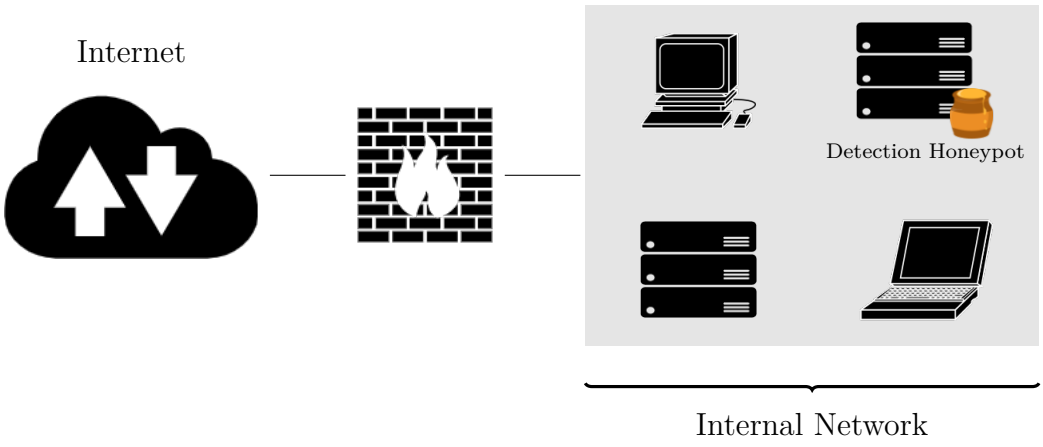


FIGURE 1.2.2
Typical deployment for detection honeypot, adapted from [5]. Note that the internal network can be arranged in several ways, namely by using a DMZ. In that case the honeypot can be placed either alongside the workstations of the internal network or in the DMZ, as illustrated in [3, 32, 36]

1.3 Interactivity

So far, a wide variety of honeypots has been defined and it should now be clear that this security element can come in many forms. Choosing among these the type that suits the best depends on the expected results for the deployment of a honeypot. One of the key metrics helping distinguish honeypots is the interactivity[5], or level of interaction[3].

In computer science, the interactivity is defined in Oxford dictionaries as

> *(Computing) The fact of allowing information to be passed continuously and in both directions between a computer or other device and the person who uses it.*

Regarding honeypots, the interactivity will therefore be their capacity to interact with their users which are members of the black hat community as defined above.

This definition simply translates the fact that a honeypot should be able to ensure the basic functionality of the legitimate system that it imitates. For instance if the honeypot responds to a port scan on 22, then it should responds to an `SSH SSH_MSG_KEXINIT` message coming to this port.

C. Sanders categorizes in his book[5] honeypot level's of interaction based on the amount of intelligence and effort needed to make it appear as a legitimate service. The relation between intelligence and effort according to interactivity is illustrated in FIGURE 1.3.1. In this graph, it can be observed that the more interactivity is needed, the less intelligent the honeypot will have to be to mimic all desired features. However, the higher the level of interaction, the more effort needs to be deployed to maintain and secure the honeypot. This is perfectly logical since if one covers more scenarios likely to attract attacks, more functionalities will increment the service appealing them, each of them being potentially usable in a malicious way.

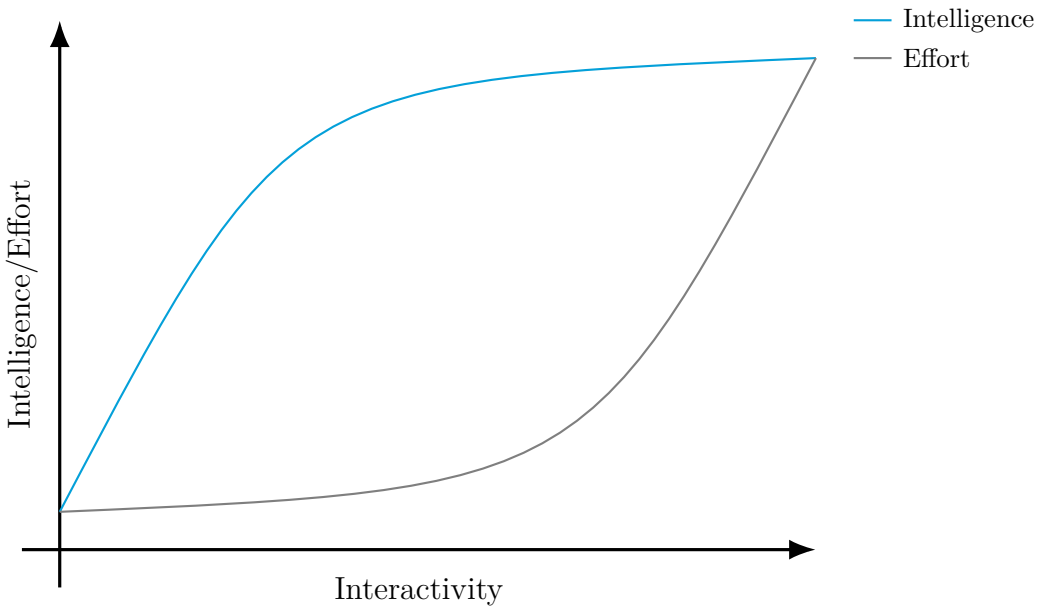


FIGURE 1.3.1
Relationship between intelligence and effort level needed to achieve an amount of interactivity through a honeypot.[5]

From the interactivity definition, one can therefore categorize honeypots based on this metric along a spectrum ranging from low to high. One of the important things to understand is that this range is rather continuous than discrete. The classification of honeypots based on interaction level is therefore subjective but is a good indicator of resources needed to maintain the device.

1.3.1 Low Interaction Honeypots

Low interaction honeypots emulate one or more systems, services or tokens without offering full OS functionalities to the attacker. Generally, this type of honeypot mimics specific services like HTTP, IP, etc. The level of interaction of these devices is very limited. They therefore implement very small amount of responses to stimuli, which limits attackers' field of action.[3, 5]

These honeypots are generally used as detection honeypots. They are easy to design, deploy and maintain given their simplicity. Since the interactivity is limited, the risk associated to these devices is also the lowest. Cisco ASA honeypot[39], Honeyd[40], Specter[41] are examples among many others of low interaction honeypots.

1.3.2 Medium Interaction Honeypots

Medium interaction honeypots are between low and high level ones and try to take advantages of both these constructions. They provide a wider range of features than low interactivity honeypots while still not offering all the functionalities of OSes. For instance, a virtual machine (VM) configured as a honeypot can be considered as medium level interaction.[3] While offering similar functionalities as the true OS to the user, virtualized operating systems do not access the hardware directly but through the virtual machine manager.[42]

Since medium interaction level honeypots have a greater complexity than low level ones, the risk associated to them also increase. However, using these technologies allows to collect more contextual information, which is not limited to transactional information (***i.e.*** data collected about the circumstance of the attack but not the attack itself) anymore. Deploying such devices requires serious security considerations in order to avoid misuses of the interactivity added. Some examples of medium level interaction honeypots are Cowrie[43], Miniprint[44] or Sticky elephant[45].

1.3.3 High Interaction Honeypots

High interaction honeypots offers all functionalities provided by the operating systems. They are complete machines left in attacker's hands deployed inside the infrastructure providing the same level of interactivity as a legitimate hardware asset in the organization. To avoid their harmful usage toward other devices in a network, they are most often placed in a isolated environment such as behind firewalls.[3]

This type of honeypots offer a more significant potential to capture and understand security threats which makes them ideal candidates for research honeypots. However, they require more skills and resources to deploy, configure and maintain. Regarding the wide range of functionalities they offer to an attacker taking their control, they represent a high risk in the security plan and can be used as powerful weapons. Capture HPC[46], Sebek[47] or Argos[48] are examples of high interaction honeypots.

1.4 Conclusion

In this chapter, the taxonomy of existing honeypot technologies has been defined as well as the interactivity, which is a key metrics when designing such a device. Defining the goal sought while deploying a honeypot is mandatory since, as shown in the previous sections, this will trivially define the type of honeypot to use.

Logging and Monitoring Honeypots

Honeypots and their various types being now defined, it is time to address one of the biggest consideration of the building process of such a device: Logging and Monitoring techniques. Indeed, whatever the purpose of the built honeypot, its primary goal is to collect data about abnormal behavior appearing inside an infrastructure. The tools used to trigger such events and inform the incident response team need to be chosen carefully, first to be sure that the proper information is collected but also to ensure that the right person received it properly. In this chapter, some definitions will be given to standardize knowledge. Then a small discussion about alerting metrics will be provided. After which, a L&M framework for honeypot will be introduced and finally, Cisco Secure Endpoint (formally Cisco AMP for Endpoint), the tool used to monitor the honeypot created during this Master's thesis will be presented.

2.1 Definitions

2.1.1 Events

The National Institute of Standards and Technology (NIST) adopted in 2012 the following definition of an information technology (IT) event

> *An event is any observable occurrence in a system or network.*[49]

A multitude of event examples should therefore come to the reader's mind such as user login, email being received or sent, new DHCP configuration done for an end host,...

2.1.2 Incidents

A differentiation should be done between simple events and incident inside an IT infrastructure. Still coming from the NIST, one can adopt the following definition

> *An incident is a violation or imminent threat of violation of security policies, acceptable use policies, or standard security practices.*[49]

An incident is therefore always an event while an event is not necessarily an incident. One can consider incidents as events having a negative impact on an infrastructure. Examples of incidents among others are unauthorized access to a system or a software, huge amount of network traffic directed through a device preventing it from working properly (Denial-of-Service, ***a.k.a.*** DoS, attack), exposition of a confidential file to the public Internet,...

2.1.3 Logs

In a computing context, a log is defined as

> *The automatically produced and time-stamped documentation of a hardware or software event initiated by a person or a running process relevant to a particular system.*

Every operating system keeps logs in log files containing records of the events having occurred on a particular machine. The logs format may vary depending of the logging system generating them.

For instance, CODE 2.1 shows the log generated by macOS Monterey operating system for an authentication failure on a device (for which UUID has been anonymized). These logs can be read either from macOS' Console application or by using the `log` utilities from the terminal. One can see from this example a real life log following the general structure made of a timestamp, some contextual information (***e.g.*** network basic information, parameters of the device initiating the log, ...) and a small message summarizing the event.

```
1 2022-01-27 17:00:23.819305+0100  localhost opendirectoryd[133]: (
    PlistFile) [com.apple.opendirectoryd:auth] Authentication failed for
    <private> (XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX):
    ODErrorCredentialsInvalid
2 2022-01-27 17:00:27.083917+0100  localhost opendirectoryd[133]: (
    PlistFile) [com.apple.opendirectoryd:auth] Authentication failed for
    <private> (XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX):
    ODErrorCredentialsInvalid
```

CODE 2.1

Apple's authentication attempt failure log. The UUIDs used in these logs have been anonimized.

2.1.4 Alerts

Alerts are defined in this thesis as

> *A machine-to-person communication acknowledging that a particular event occurs.*

Every alert is in fact a log while the contrary is not true.[5] Indeed, when logs meet a certain condition, one may decide that the team responsible for this event should be informed in an active manner, meaning that a notification will be directly sent to this department. Typically, an alert is sent using the communication application preferred by the organization running the asset that will initiate the notification (***e.g.***, Webex, Slack, emails, ...).

As an example, consider a company using a few dozen virtual machines to host an application running databases. The disk utilization of one of these VM reaches 90%. It is a safe bet that no operation team member will verify each machine memory state every day to ensure that none of these are full. To avoid bad surprises regarding memory usage, it could be a good idea for this company to deploy a monitoring and alerting toolkit that will trigger on a certain level of memory usage and alert the operation team. For instance, if the OS ran by each of these VMs allows it, the company can choose to use the

Prometheus[50] toolkit in conjunction with Slack as a communication application. FIGURE 2.1.1 shows a typical alert generated by Prometheus as a Slack message.

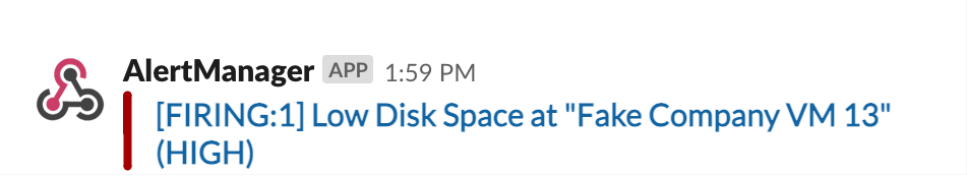


FIGURE 2.1.1
Example of an alert mentioning a high disk volume usage initiated by the Prometheus toolkit and using Slack as communication application.

Notice that alerts come in various forms like pop-ups, emails, text messages, ... The key point is that an alert is a communication from a machine to a human for an event raising a particular condition.

2.2 Honeypots Alerting Metrics

Now that the key terms used in this chapter have been defined, it is time to discuss the efficiency of an alerting mechanism of a honeypot. In general, the efficiency of such a system can be easily evaluated using two quantities

1. **False positive:** the amount of mislabeled security alerts ***i.e.***, the number of security alerts received while no security incident occurred.
2. **False negative:** the amount of uncaught security threats ***i.e.***, the number of security alerts not triggered while a security incident occurred.

When designed properly, a honeypot should be characterized with both low false positive and false negative alert rate. Indeed, anything happening inside a honeypot results in an incident[1] since honeypots are not supposed to be used for legitimate tasks inside an infrastructure (***i.e.***, to have production activity) and not to be accessed by anyone except the administrator of this device. This consideration means that any activity generated by a honeypot must normally be malicious as long as the monitoring system has been correctly configured.[5, 6, 27, 32, 33, 51]

Since information collected in a honeypot generate no (or a few in worst cases) false-positive/negative, the noise ratio in these is very low making them high value data. This means that any material collected by a honeypot should always be investigated. Furthermore, the low occurrence of false-positive leads to rapid detection of legitimate threats. Also, since any interaction with a honeypot should be considered as a legitimate threat, unknown attacks will be found just as quickly as known ones.[27, 32]

2.3 Monitoring Framework

All those who have worked with software will agree on the fact that properly monitoring an application behavior is not an easy thing. The data provided by application logs

[1] This obviously excludes the noisy legitimate activity of the machine (***e.g.*** automatic update, monitoring scans, Address Resolution Protocol (ARP) traffic, ...).

are essential to both development and IT operations teams to detect and diagnose undesired behavior. However, even if many technologies facilitating the logging of assets exist, monitoring complex systems and getting valuable data from the generated logs remains challenging.[23]

In his book *Honeypots for Windows*, R. A. Grimes[27] introduced the honeypot monitoring framework depicted in FIGURE 2.3.1. Although being very general, this theoretical strategy remains a good reference in terms of monitoring system deployment for assets. This is why it was chosen for this thesis.

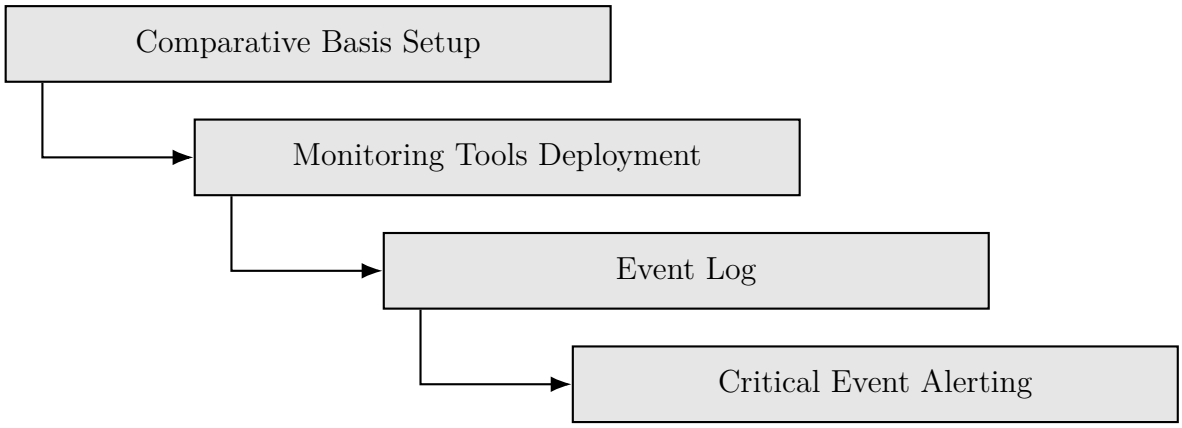


FIGURE 2.3.1
General service monitoring framework presented in the as part of the honeypot technology.[27] This framework has been kept general and can therefore be easily transposed to any system monitoring implementation.

In the following sections, each step of the presented framework (***cf.*** FIGURE 2.3.1) will be briefly reviewed as well as the challenges for each of them.

2.3.1 Comparative Basis Setup

To be able to monitor abnormal system activity, the very first step is to define the normal one, ***i.e.*** the representative behavior of the uncompromised system. Indeed, every system will have a minimal activity being made, for instance of automatic software updates, clock synchronizations, basic network protocol traffic,... This elemental workload needs to be carefully measured and documented otherwise the monitoring system could generate noisy logs and alerts, triggering these activities as being abnormal, as mentioned in section 2.2.

The activity of a system although being very important to detect abnormal behaviors, another information should also be documented with utmost care: the system initial configuration. Indeed, even if a change in the configuration of an architecture should normally create activity, finding out what has been manipulated or changed may turn out being a tedious task. If the initial state of a device is properly documented, comparing it with its state after an event or an incident will ease the threat analysis.

Nowadays, a lot of powerful tools help to collect information about both system activity and configuration. Most OSes offer integrated backup managers and softwares like Network Monitor, Netstat,... allowing to easily capture, document and analyze a machine state for instance. Living in Cloud's age, snapshots of virtual devices can be effortlessly

and quickly made. Depending of the deployed infrastructure, a whole panoply of tools exist to ease the documentation of systems' normal behaviors. The Cisco Secure portfolio, regrouping all the monitoring tools used in this thesis, notably offers such features as discussed in section 2.5.

2.3.2 Monitoring Tools Deployment

Once the comparative basis created, the second step in the introduction of the monitoring framework is to define a monitoring plan. The latter should define how the system should be monitored.

Once again, a lot of tools exist to monitor an IT infrastructure from commercial keyloggers (***e.g.*** Spyrix, Kidinspector, etc.) to network analyzer (***e.g.*** Solarwinds, NetSpot, etc.). The chosen tools to monitor any system, including honeypots, will of course highly depend on its purpose. For instance, it would make no sense to use a monitoring system conceived for Windows inside an infrastructure only running Debian OS. Or to use a wireless networks analyzer to detect a local code excecution. Tools must be chosen carefully and their functionalities as well as their limits must be known to ensure the proper data collection and the safety of the deployment.

When speaking of monitoring system, two techniques of deployments can be found, called in-band and out-of-band monitoring.[27] In-band monitoring functions inside the monitored system and generally requires it to be fully functional in order to collect data. Out-of-band monitoring systems operate outside the confines of the system and can capture information even if the latter is down. Some examples for the first category are Windows event logs, keystroke logging,... while the second one can be achieved by intrusion detection systems, packet-capturing utilities,... Both techniques have of course their own advantages and drawbacks. For instance, if the payload of a network packet sent to a device is encrypted, only in-band monitoring has a chance to record it in plain text. On the other hand, out-of-band monitoring data are less susceptible to be compromised.[2] Once again, using one, the other or both technique(s) will depend of the use-case in which the monitoring framework is applied.

C. Sanders in *Intrusion Detection Honeypots: Detection Through Deception*[5] interestingly highlights the fact that the monitoring system should not be excluded from the monitoring scope. Indeed, monitoring systems remain part of the infrastructure being monitored and should therefore be watched as well.

The protection of the monitoring communications is one final consideration that should be taken into account while defining the tools used to monitor a system. Indeed, both in-band and out-of-band monitoring generally report their data to an external management system. Both R. A. Grimes and C. Sanders agree that monitoring system information should be protected by encryption in transit and access control.[5, 27] These measures should not only prevent eavesdropping of these sensitive information but also ensure their reliability.

[2]Think of an attack to a device. Most of the time, hackers erase the clues that could lead back to them or to the discover of a security threat. If planned correctly, it can be nearly impossible for hackers to even know their activities are being recorded through an out-of-band monitoring system and therefore to cover his/her tracks.

Once again, among the tools offered by the Cisco Secure portfolio, several are dedicated to either in-band or out-of-band system monitoring as explained in section 2.5.

2.3.3 Event Log

Now that *how* monitoring is performed has been defined, the next and most difficult step of the framework is to define *what* should be monitored. Indeed, monitoring tools will gather a lot of data but defining which of them are of real interest for the monitored system is crucial for their proper configuration. For each event designated as having to be monitored, an event log will be generated by the monitoring system.

Event logs normally exist to provide contextual information about events having occurred regarding an asset of an infrastructure. However, the logging framework itself is highly dependent on the developer habits[23–26] despite the fact that logs will help way more people than developers (***e.g.*** customers, incident response teams, operations engineers, ...). There is unfortunately no magic recipe when it comes to logging in the computer world. In 2005, R. A. Grimes already mentioned that even if logging events seems simple, many people do it badly.[27] In a more recent article[23], J. Cândido *et al.* showed that logging is still an active field of research attracting researchers but also practitioners. Currently, there exists no standard logging guidelines for developers[23, 25, 26] which often complicates the analysis of the logs produced by a monitoring system. The key to produce logs efficiently is to find the balance between logging enough information to detect and define critical events while not polluting it with unnecessary data. Too often, log files are made of bursts of useless events, used mostly for debugging, giving a hard time to those who seek to analyze them.

Another problem one will be faced when defining the logging infrastructure is the lack of standardization between monitoring tools. Indeed, as for the logging guidelines for the developer, there exist no guidelines regarding how application should format a log. If several monitoring tools have been chosen in the step presented in section 2.3.2, it is a safe bet that the format of the produced logs will be different for each of them. Therefore, one can easily see that the more monitoring applications chosen, the more complex the log information. Nowadays, there exists some tools allowing to perform log formatting ***i.e.*** to extract meaningful information from general log data and standardize their formats. Among many other such tools, one can find Logstash, Sematext Logs or Splunk.

As the reader must have realized, defining clear guidelines for event logs generation is not an easy thing. As already mentioned, years of research in this area have still not made it possible to standardize logging practices.[23, 25] However, several advices can come from experience. First, to reduce the complexity of the generated log data, the number of different monitoring systems should be reduced as much as possible. Second, event logging should only produce interesting information, which require deep understanding of the monitored system. Finally, the format of the logs should ideally remain as standard as possible within the monitoring system. Using an aggregation and log format tool is a good idea when the number of assets inside the monitored infrastructure grows.

As shall be showed in section 2.5, not only do the tools from Cisco Secure offer a lot of monitoring capabilities but in addition all this information can be gathered inside a single platform, SecureX. However, logging the proper information will still remain part of the tools configuration, and therefore dependent on human free will.

2.3.4 Critical Event Alerting

Once the previous steps of the framework are properly configured, the monitoring system should have generated one or several log files[3] representing historical reports of the monitored system state and activity. The last step of the presented framework is to define, among the generated logs, which ones will raise an alert.

Recall from section 2.1.4 that an alert is triggered by a particular event and transmitted to a person. Recall also that every alert is in fact a log, therefore the same considerations as those made in the previous section stand. As for logs, alerts are defined by humans and will therefore depend of the level of criticality assessment of the event to which it relates. Once again, there is consequently no standardized guidelines to define alerts. One thing to keep in mind while configuring them is that they will involve people, it is thus important to know which event deserves to disturb someone. One thing to absolutely avoid is a *boy who cried wolf*[52] situation. This is the result of a disturbing false alert that is raised several time and ended up being ignored after a while, although being important. Hence, not only should the criticality of the alerted event be considered but also its impact on the people receiving it.

Last but not least, the communication channel must also be carefully defined while speaking of alerts. Most companies have their own internal/external communication policies, using applications designed for this purpose, ***e.g.*** emails, Short Message Service (SMS) or others. Even if the messaging software ecosystem is broad nowadays, preferring the use of the application used as alert transmission mean sounds like a good idea. Indeed, as for monitoring systems, adding new software to an infrastructure increases its complexity and trying to reduce this as much as possible is a good way to ensure its proper use. Finally, it is important to ensure that the communication channel used to transmit an alert is reliable. Alerts inform people of important events or incidents having occurred within the infrastructure. Furthermore, as already seen in section 2.1.4, alert messages can also contain sensitive information relative to the context in which they occurred inside the infrastructure (***e.g.*** IP addresses, protocol used, devices names, user account, etc.). The means of communication must therefore be trusted to avoid the interception or corruption of an alert.

2.4 Security and Honeypot Context

The presented framework in sections 2.3.1-2.3.4 has been intentionally kept general in order to introduce a monitoring methodology. Even though monitoring is highly used for security purpose, its application is not limited to this field. For instance, one may also consider application monitoring often used for bug tracking or network monitoring for failure detection. Among the huge quantity of monitored events in an infrastructure, defining which ones fall under security is not always an easy task. However, following the introduced guidelines to implement each necessary monitoring framework can help to homogenize the monitoring infrastructure and greatly ease this process.

When monitoring honeypots, which are therefore subject to security considerations, event logging and alerting becomes way simpler than for other devices. Indeed, as al-

[3] Recall from section 2.1.3 that logs are typically stored in log files.

ready mentioned in chapter 1 and section 2.1.4, whatever the honeypot type used, every event happening in such a device represents a security incident that should be analyzed. Therefore, everything happening in a honeypot will be logged without generating noisy information. One common misconception about honeypots is that they represent a high security risk when deployed in an infrastructure, internally or externally. However, a honeypot is highly monitored and if the monitoring system has been configured properly, an alert will be sent soon as an interaction with it is detected, therefore reducing considerably the risk associated to the device.[5]

2.5 Cisco Secure

Cisco Secure is the largest enterprise cybersecurity company in the world[53], offering security solutions namely for extended detection and response (XDR). XDR is a new security approach abandoning the old fashioned silo vision of security in favor of a centralized view-point on the entirety of the security of an infrastructure. All of the solutions offered by Cisco Secure can be combined together to create a powerful monitoring system for any infrastructure. Cisco Secure's portfolio contains a lot of tools, despite only several of them have been used for this master thesis. In the next sections, the solutions of interest for this writing will be presented as well as the way they interact with each other. However, it is interesting to know that the suite of tools offered by Cisco Secure is not limited to these and can cover many other features related to the security of an infrastructure. FIGURE 2.5.1 gives a graphical representation of the tools that will be presented below and the way they interconnect.

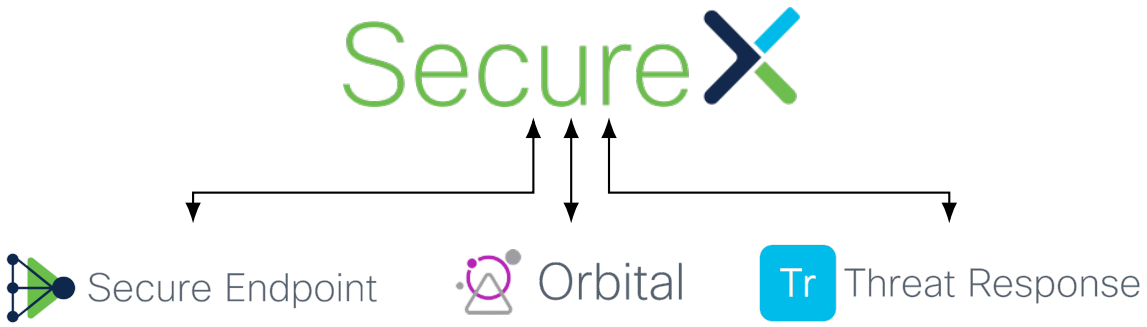


FIGURE 2.5.1
Cisco Secure tools used in this master thesis and their relationships. SecureX is the platform centralizing the access to Secure Endpoint, Orbital and Threat Response.

2.5.1 Cisco SecureX

SecureX is the central tool of Cisco Secure's XDR approach. It is an integrated platform centralizing security information of an infrastructure, allowing connectivity of already existing security solutions and offering the possibility of automatizing security workflows.[53] By using this platform, security operators have a global view on their infrastructure from all the tools they use and can use *Orchestration* to implement easily organization security workflows executing on their systems.

2.5.2 Cisco Secure Endpoint

Cisco Secure Endpoint implements the Endpoint Detection and Response (EDR) approach giving an overview of each endpoint security state. Secure Endpoint visibility is therefore limited to the endpoint on which it runs. It offers threat hunting capabilities, using file analysis to detect malicious content but also some incident response functionalities such as the device quarantine or the blacklisting of application/IP addresses. While Cisco SecureX offers a global overview of the security infrastructure in place, Secure Endpoint focus on the endpoint health only. However, the computer park made of the devices running Secure Endpoint is still manageable using a global dashboard presenting its general metrics.

2.5.3 Cisco Orbital

Cisco Orbital is based on Osquery[54], implementing an SQL style model allowing to query a device to get information about its state. The whole device is therefore exposed as a relational database from which information can be retrieved by executing SQL queries. The complete schema of the tables managed by Osquery can be found on this software web page.[54] Even if most of them are supported in Orbital, those which are excluded are mentioned in the documentation of Orbital.[55]

Orbital's job is therefore to ease information collection of a machine within the limits of the ones contained in the rational schema of Orbital. This is thus once again a device centric tool allowing to get information about a device state.

2.5.4 Cisco Threat Response

Cisco Threat Response is a security investigation and incident response application being part of the SecureX toolkit. This application propose a GUI to simplify threat hunting and incident response by accelerating detection, investigation, and remediation of threats. Through graphs and pre-sorted data, Treat Response help to visualize the context of potential incidents having taken place in the managed infrastructure and eases their judgment and archives.

2.6 Conclusion

In this chapter a strong framework has been defined, allowing to overcome the difficulties that may arise while implementing poor monitoring strategies. This framework has afterwards been presented for the particular case of honeypots. Finally, the monitoring toolkit being part of the Cisco Secure portfolio used in this thesis was introduced.

Honeytokens

In today's world, even if some cyber criminals are driven by other causes, the principal reason pushing cyber attacks remains money.[56, 57] Money can be earned from different computer attacks, for example renting infected machines such as botnets. Computer resource renting can be a business for sure, however another more lucrative one targets the market of data. Clive Robert Humby, a British mathematician, said in 2006: *Data is the new oil*, and the black hat community as well as a lot of respectable companies clearly understood that. This is why most companies heavily invest in security plans, attempting to ensure confidentiality, integrity and availability of data inside their IT assets.[58]

In this chapter are presented honeytokens, which take advantage of the well known high value of data to generate interest for an asset within the black hat community. As usual, a formal definition will be stated for these particular resources. After what, their properties as well as their different types will be introduced.

3.1 Definitions

In this thesis, the definition of honeytoken adopted is the following.

> *Honeytokens are security resources mimicking valid data placed in a system or a network under the form of complete files, entries within files, or special records.*[5, 30]

From the definition given in section 1.1, honeypots can be found in the form of any computers resources which implies computer systems but also assets in many other forms. This is actually one misconception when speaking of honeypot that should be clarified with the introduction of the honeytokens concept, term proposed by A. Paes de Barros during an email exchange with L. Spitzner[59], being actually a subset of honeypot.[5, 30, 34, 60] In fact, in his series of honeypot papers, L. Spitzner gave the following definition for these resources

> *A honeytoken is a honeypot that is not a computer.*[60]

This definition highlights what has been discussed supra, however it has been judged not precise enough in the context of this thesis.

3.2 Properties

To ensure the interest of honeytokens, several properties have been defined by B. Bowen *et al.* in 2009.[61] Due to the completeness of the definitions of each of them in this article, they will simply be cited in this master thesis without elaborating unduly. Therefore, to be efficient, a honeytoken (***a.k.a.*** decoy in [61]) should follow the following properties.

Property 3.1. **Be believable:** a honeytoken should be difficult to classify as non legitimate information[1], it should look valid. From the definition of believability has been introduced brilliantly the concept of *perfect honeytoken*, inspired by the one of *perfect secrecy* proposed in the cryptography community.[62] Basically, the perfect honeytoken is the one chosen with 50% probability as being authentic information when presented aside a real piece of data.

Property 3.2. **Be enticing:** honeytokens should arouse desires, be attractive for the any illegitimate user finding them. This will of course be highly dependent of insider's intents or preferences which are not easily measurable. However, some obvious type of data having black market value such as credentials, personal or bank information among others naturally seem to be good candidates from which to draw inspiration while creating honeytokens.[56, 57, 63]

Property 3.3. **Be conspicuous:** honeytokens should be easily discovered. While enticing honeytokens are chosen because of the interest it arouses, conspicuous ones are targeted because they are easily found. For example, a completely common document placed as an icon in the desktop environment of a computer is more likely to be consulted than if it is hidden within a complex folder tree.

Property 3.4. **Be detectable:** this property ensures that the exploitation of honeytokens must be detected. For instance, if one creates information relative to a fake company which is leaked over the Internet, since this corporation is totally fictitious, one can conclude of the corruption of the whole system (or at least, one of the systems) where the information was placed.

Property 3.5. **Ensure variability:** a honeytoken should not be easily identifiable due to some common invariant or redundant pattern. This ensures that no signature allows the black hat community to systematically detect a generated honeytoken.

Property 3.6. **Ensure non-interference:** while deploying honeytoken in an operational environment, not specially conceived to only contains such data, they can potentially interfere with the system. For instance, if properly designed, not only an insider can be lured by a honeytoken but also users. Taking as an example a password file as honeytoken, imagine if a fooled user stored its real credentials inside this file designed specially to be illegitimately used. The more enticing or believable a honeytoken is, the more likely it will lead users to confuse it with a legitimate document.

Property 3.7. **Be differentiable:** this property translate the fact that legitimate users should be able to differentiate honeytokens from actual data to avoid interference, while an insider should not.[13]

Properties 3.1-3.7 should serve as guidelines to develop proper honeytokens. However, even if considering all of them is always a good idea to create a functional system, depend-

[1] Even if by definition, honeytokens are non legitimate information.

ing on the context, some may be more important then others. For instance, if the system that will contain the honeytoken is not used by any legitimate user, it seems trivial to consider property 3.1 more important than 3.6. Indeed, in such a scenario, since no legitimate user should encounter the created honeytokens, the risk of him or her considering them as actual data is non-existing. However, since only insiders should face these honey data, making them completely believable is mandatory for making the lure complete.

3.3 Types of Honeytokens

As the reader should have understood with the two previous sections, honeytokens have extensive flexibility and their diversity is only limited by their creator's imagination.[60] In 2020, C. Sanders[5] presented the honeytoken taxonomy by discussing several types of these resources (honey documents, honey files, honey folders and honey credentials). Even if working by example is great to understand general concepts easily, it does not define a strong theoretical basis guiding the definition of honeytokens.

The honeytoken classification scheme used in this thesis has been proposed a bit earlier. In 2020, S. Srinivasa *et al.* introduced in the article "Towards systematic honeytoken fingerprinting" a systematic honeytoken classification based on the infrastructure level where they operate to examine honeytoken-specific fingerprinting.[30] The following sections define the level of operation as well as the honeytoken type that can be found within them.

3.3.1 Network Level

In an infrastructure, network wide resources are the ones shared between several network instances. This straightforward vision logically follows the well-known concept of computer networking.

Network honeytokens are therefore the ones replicating a network resource, ***i.e.*** data appearing as legitimate for any exchange between two numeric devices. For instance, fake Domain Name System (DNS) entries can be considered as network honeytokens. Indeed, DNS names are resolved while searching for a service on the web, typically by querying DNS servers. If a honeytoken under the form of a DNS entry is created inside a DNS server, following the definition of honeypots from which honeytokens derive, it should not be resolved for any legitimate usage. Because the DNS name resolution is active network wide and not on a single device, such honeytokens can be classified as part of the network level ones.

3.3.2 File Level

When limiting the scope to the file level of an infrastructure, the focus is on a single numeric file being a collection of related information defined by its creator.[42] Generally, files are strictly formatted following standards (***e.g.*** Portable Document Format, Word documents, Portable Network Graphics, ...).

File level honeytokens are therefore entire computer files that can be found in file systems, not just pieces of data within legitimate files or devices. For example, any `.docx` document respecting the properties introduced in section 3.2 containing only unreal data

fall under this category of honeytokens.

3.3.3 System Level

The system level, probably being the most abstract level defined in this section, considers resources confined within a single device while not representing a file.

Honeytokens active in the system level therefore rely on system components to simulate one of its ressource. For instance, directories (***a.k.a.*** folders) are part of a system while not being files. A system level honeytoken can therefore be one simulating a whole folder within a device, or even a folder tree.

3.3.4 Data Level

The data level refers to the smallest unit of information that can be simulated by a honeytoken. While such type of honeytoken can be part of file, system or network level ones, they do not need a whole context to be efficient.

For instance, think about an email address designed to be a honeytoken. This email address should not receive any email except if it is leaked. As if, this represents a data level honeytoken and if some abnormal activity related to it is detected, it is due to a data leak from the place where it was stored. Of course, such data can be easily embedded into system level honeytoken (***e.g.*** by setting the device owner address to this one), file level ones (***e.g.*** by adding this email address into an Excel file listing company employees) or even at the network level (***e.g.*** by setting the address to the contact one for a honey website).

3.3.4.1 Honey Credentials

Honey credentials are a type of honeytoken falling under the data level family often referenced in the literature, which is why a paragraph mentioning them seemed appropriate. C. Sanders define them as being fake credentials designed to lure attackers attempting authentication to a service.[5]

Nowadays, credentials remain one of the most leaked type of data[56, 57] while also having the longest average time of compromise identification.[63] The value of honey credentials lies in their simplicity (regarding both creation and deployment) but also in their high efficiency to detect credentials leak. Indeed, one may imagine how easy it is to create a web account and monitor any access to it in today's Internet world. As an example, consider generating a Google mail account configured to trigger any connection to it either via a multi-factor authentication (MFA) or using an alert to another email address. Placing these fake credentials into a list of legitimate ones can allow to easily detect a leakage of this list. If any connection notification of the usage of the created honey credential is generated, one can be sure that the file containing the data has in fact leaked. This simple use case shows how powerful honeytokens can easily protect infrastructures.

3.4 Conclusion

In this chapter, honeytokens have been presented, which are a subcategory of honeypots mimicking legitimate infrastructure resources and not systems. Through the defi-

nition of properties and types of honeypots, general guidelines have been introduced to create such security resources efficiently and correctly. The numerous examples given in the sections supra should suffice to convince the reader of the elegant simplicity as well as the high efficiency of honeytokens. If it is not the case, section 5.5.3 from part II will provide another usage of honeytoken inside a whole deployment.

Part II

Practical Implementation

A First Remote Desktop Protocol Honeypot as a Proof of Concept

In this chapter, the methodology of the first honeypot deployment made in this master thesis will be reviewed. The motivations behind it, the chosen emulated service, its particular configuration as well as the monitoring strategy used will be elaborated in the next sections. Finally, the loop shall be closed with the review of each characteristics of the created honeypots as given in part I of this document.

4.1 Motivations

With this first deployment, the interest of using the Cisco SecureX tool to monitor honeypot devices should be underlined. The idea behind this so called proof of concept (PoC) is to simulate a well known service inside a honeypot to attract illegitimate traffic.

Nowadays, a significant proportion of users still opt for Windows as operating system for their personal computer[64] but also when working with web servers[65]. Clearly, when trying to simulate a real life system behavior to design a honeypot, Windows is a good starting point due to its broad usage. The very first technical decision is therefore to use a Windows host to create the honeypot, which is perfectly compatible with all the SecureX tools used during this Master's thesis.[55, 66, 67]

Even if using a Windows device is already a strategic decision, if this device remain completely inactive, nothing very interesting can be seen within it, resulting in a poor design of the honeypot. The idea is therefore to enable a service in the Windows honeypot, interesting enough to attract illegitimate traffic. Since this first deployment is a PoC, another requirement is to keep the emulated service as simple as possible. A good way to attract traffic to a device is to mimic a well know network service provided by the created honeypot (***e.g.*** HTTPS, SSH, FTP, SMTP,...). Nowadays, a lot of attacks are indeed orchestrated remotely, using networks capabilities to interact with a device.[56, 57, 68] It is therefore a safe bet that opening a network port on a device should attract traffic, provided of course that it is placed in the right place.

A super easy way to make a Windows device accessible from the Internet is to use Windows remote desktop protocol (RDP). RDP allows taking control of a computer remotely and provides a graphical user interface (GUI) making it frequently used by system administrators for remote troubleshooting and maintenance.[5] Once the right credentials are given to initiate a RDP connection, the user is granted the access rights correspond-

ing to this profile. As said before, this protocol is broadly used by system administrators probably owning administrator rights on the machine they remotly access. By putting on a black hat[1] for a moment, the interest behind the use of such a protocol to control a complete machine is quite straightforward. One can just have a look at the price of renting botnets on the current market to be convinced of this.[70]

As explained in chapter 2, one of the key consideration to make while designing honeypots is how they will be monitored. Once again, since this PoC (and this master thesis in general) serves to demonstrate the interest of using the XDR concept and especially the one implemented by SecureX, of course the tool suite offered by this platform is used to monitor this first honeypot. Section 4.4 should highlight how using this technology is indeed a good choice.

Based on all these motivations, the first honeypot demonstrating the monitoring capabilities of SecureX for such a device will therefore be a Windows machine with the remote desktop protocol enabled. With this, the three main goals considered should be achieved, being

- Attract analyzable traffic on the honeypot;
- Use SecureX to detect this traffic and collect information about it;
- Keep the honeypot as simple as possible.

4.2 Remote Desktop Protocol

Remote desktop protocol is a Windows proprietary protocol providing a graphical user interface (GUI) to remotely control a Windows device. RDP is an extension of the T.128[71] (Multipoint application sharing) protocol standardized by the ITU Telecommunication Standardization Sector (ITU-T). It also relies on two other standards of this family being the T.122[72] and T.125[73] ones. FIGURE 4.2.1 shows the various standard over which RDP runs and underlines their place in the TCP/IP model. In this figure, TPKT, ***a.k.a.*** the ISO transport service on top of TCP[74], allows peers to exchange transport protocol data units (TPDU). The X.224[75] standard is a connection-oriented transport protocol providing a connection-mode transport service. It is used by RDP during the initial connection request and response. Finally, the T.125 standard, ***a.k.a.*** multipoint communication service protocol, allows RDP to run over multiple communication channels.

In the blog post *Explain Like I'm 5: Remote Desktop Protocol (RDP)*, S. Reiner gives a detailed overview of the technical capabilities of RDP, going beyond the scope of this thesis.[76] Knowing how a simulated service inside a honeypot works is essential in order to be able to differentiate the normal activity of the device from the abnormal one and therefore to configure properly the event log step of the monitoring framework (***cf.*** section 2.3.3). However, this deployment is not intended to study the slightest flaw within Windows Remote Desktop Protocol. Therefore, the knowledge regarding RDP's technical capabilities can be limited to what has been presented supra as well as the more operational consideration described in section 4.3.

[1] In reference to the black hat[69] community.

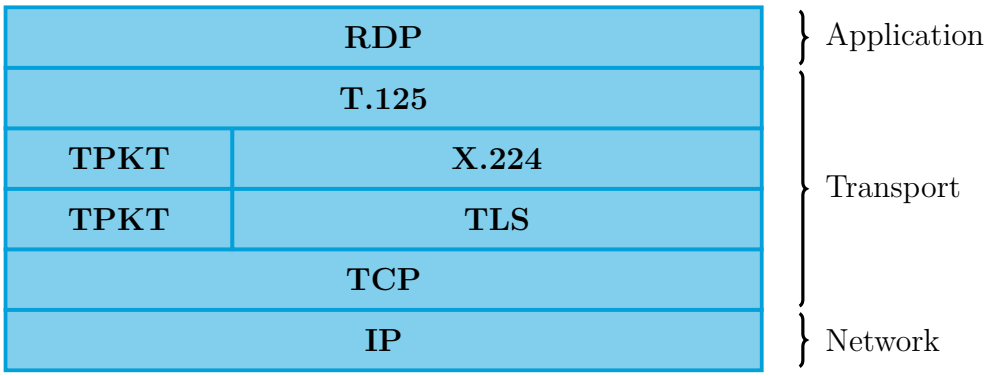


FIGURE 4.2.1

Remote desktop protocol stack of used well known standards. TPKT[74] allows peers to exchange transport protocol data units (TPDU). The X.224[75] standard is a connection-oriented transport protocol providing a connection-mode transport service used by RDP during the initial connection request and response. The T.125 standard, allows RDP to run over multiple communication channels.Adapted from [76].

4.3 Creating a RDP Honeypot

To ensure security, sometimes one can decide to implement a fake service inside a honeypot, offering the functionalities of the legitimate simulated one. For instance, OpenCanary[77] is a honeypot that can simulate different HTTP login pages while actually, these connections page lead nowhere. They just record the credentials given as input to the form in the web page and alert the honeypot administrator by mail of the attempt of corruption.

However, in some cases, the features offered by legitimate services suffice to actually completely secure their usage inside a honeypot. That is the case with RDP and this approach has been chosen here, still keeping in mind the fact that this proof of concept should be as simple as possible. Indeed, once activated, RDP allows to limit the device user accounts able to actually log in from this protocol. By using the edge case of not allowing any user while enabling the service, security is ensured since nobody will be able to control the machine using RDP whatever the credentials given to the service, even if they are legitimate.

4.3.1 Enabling RDP

Enabling remote desktop on Windows is quite easy as it is an integrated feature to this OS. Notice however that a professional, enterprise or server version is required to use this service.[2] On such machines, performing the following steps will suffice enabling RDP

1. Open *Settings* from the *Start* menu;
2. Open *System*;
3. Open the *Remote Desktop* tab from the left sidebar;
4. Turn on *Enable Remote Desktop* toggle switch;
5. Click the *Confirm* button.

To check the proper configuration of RDP, one can check that the device actually listens

[2]See this website for the whole OS compatibility list.

for connections on port 3389, which is the standard port of RDP. Code 4.1 shows how to use the `netstat` utility from Windows PowerShell to check if RDP has been properly configured.

```
PS C:\Windows\system32> netstat -an | Select-String "3389"

  TCP    0.0.0.0:3389           0.0.0.0:0              LISTENING
  TCP    [::]:3389              [::]:0                 LISTENING
  UDP    0.0.0.0:3389           *:*
  UDP    [::]:3389              *:*
```

CODE 4.1
RDP configuration check using PowerShell and the `netstat` utility.

Once RDP properly enabled, one last configuration needs to be performed in order to allow RDP traffic to reach a device. Every Windows runs Windows Firewall which, by default blocks all traffic trying to initiate a connection with the device unless there is an exception rule created. Allowing RDP traffic passing trough Windows Firewall is done by performing the following steps

1. Open *Settings* from the *Start* menu;
2. Open *Update & Security*;
3. Open the *Windows Security* tab from the left sidebar and open *Firewall & network protection* from the right-hand listing;
4. Click on *Allow an app through firewall*;
5. In the new opened window *Allowed apps*, click on the *Change settings* button (ran as administrator);
6. Search for *Remote Desktop* in the *Allowed apps and features* list and select all three boxes (the *application*, the *Private* and *Public* accesses);
7. Click on *OK* to validate the Windows Firewall configuration.

Once again, the proper configuration of Windows Firewall can be checked using PowerShell as shown by Code 4.2.

```
PS C:\Users\vince> Get-NetFirewallRule -DisplayGroup 'Remote Desktop' |
    Format-Table DisplayName, Enabled, Profile, Direction

DisplayName                             Enabled Profile Direction
-----------                             ------- ------- ---------
Remote Desktop - User Mode (TCP-In)        True     Any   Inbound
Remote Desktop - User Mode (UDP-In)        True     Any   Inbound
Remote Desktop - Shadow (TCP-In)           True     Any   Inbound
```

CODE 4.2
Windows Firewall allowing RDP configuration check using PowerShell and the `Get-NetFirewallRule` command.

4.3.2 Deny Account Logon Using RDP

One way to significantly reduce the risk of unauthorized access to the created device is to not grant any user account with the privilege of login using RDP. This may seem a bit confusing since honeypots are actually designed to study abnormal activities and by

shutting the door to these, one can think that the created honeypot will lose in terms of data richness. However, remember that studying insider behaviors is the goal of one family of honeypots, referred to as the research honeypot. A second one does also exists, the production honeypots family which aims at detecting such intrusion (***cf.*** section 1.2). Furthermore, recall that here the main goal of the honeypot is to detect connection attempts using RDP to a device, no more, no less. The designed honeypot has therefore no intention of studying the credentials entered during the connection attempts, the protocol's pitfalls used to initiate a privilege escalation or whatever. Therefore, by not allowing any user to connect using RDP, the service still remain idle while the risk of illegitimate device control is significantly reduced.

Denying users or groups of users to login via RDP can be easily performed using the *Deny logon through Remote Desktop Services* privilege. Setting this privilege can be done with the following steps.

1. From Windows search bar search for *Gpedit.msc* and open *Edit group policy*;
2. Navigate through *Local Computer Policy* → *Computer Configuration* → *Windows Settings* → *Security Settings* → *Local Policies* → *User Rights Assignment*;
3. Open the *Deny logon through Remote Desktop Services* policy;
4. Click on *Add User or Group...*;
5. Click on *Advanced* from the new opened window *Select Users or Groups*;
6. Click on *Find Now*;
7. Select all users and groups in the search results and click on *OK*;
8. Click on *OK* window *Select Users or Groups*;
9. Click on *Apply* and then *OK* from the *Deny logon through Remote Desktop Services* policy window.

After performing these actions, the created honeypot runs a RDP service while not allowing anyone to use it successfully. Now that everything is operational, the monitoring strategy should be put in place in order to detect connection attempts to the device.

4.4 Monitoring the Honeypot

The monitoring strategy for this honeypot will of course rely on the monitoring framework introduced in section 2.3. The following sections will present the strategy adopted to implement each step of this framework, making sure that the honeypot is properly guarded.

4.4.1 Comparative Basis Setup

As it will be explained in section 4.5.1, the created RDP honeypot is deployed in a virtualized environment. The hypervisor (***i.e.*** VMWare ESXi 6.7.0) used to manage the virtual machines running the created RDP honey services implements a snapshot functionality, allowing to capture the whole device state as it is the case for most of nowadays hypervisors. In the machine's initial state, a snapshot of the honeypot configured by following the steps given in section 4.3 is taken as a comparative basis.

Not only will this snapshot be useful to detect change in the device in the case where it would be compromised but this will also allow to quickly and easily redeploy the fully

functional created honeypot without redoing the work of configuring it. Indeed, one of the main goals of snapshots is to allow users to restore a virtual machine state efficiently.

4.4.2 Monitoring Tools Deployment

As already mentioned, the tool used to monitor this honeypot is SecureX. Therefore, the very first step regarding monitoring is to install the Secure Endpoint connector on the honeypot.[3] As explained in section 2.5.2, Secure Endpoint implements the EDR concept and will therefore monitor activities inside the devices. Most of the events occurring in a device monitored using Secure Endpoint will be observable from the Secure Endpoint administration console. Therefore, activities taking place inside the honeypot will be investigable from Secure Endpoint, in particular through the use of Device trajectory but also Threat Response for events judged as critical.

Once Secure Endpoint installed on the honeypot, for the monitoring strategy to be fully functional, Orbital still needs to be deployed on the device as well. Once again, due to the high connectivity implemented between SecureX tools, this can be simply done using the Secure Endpoint administration console.[4] Once Orbital installed on the device, no more endpoint configuration is required. Note that even if Orbital is installed only at this step of the monitoring framework, one of its functionalities can be used to increment the comparative basis setup from section 4.4.1. Indeed, from Orbital's query catalog, the user can find the Forensic Snapshot ones. Theses queries allow to get information from most of Orbital tables and therefore about the machine state for the different OSes supported by Orbital. As the reader will see in section 4.4.3, the device state in the case of an abnormal event will be saved using this Orbital capability. Therefore, to be able to compare collected data with initial ones, it might be a good idea to add an Orbital forensics snapshot to the basis documented in section 4.4.1.

As the reader will have noticed, in this study case only in-band monitoring is implemented. Indeed, due to the placement of the honeypot inside the infrastructure, out-of-band monitoring does not seem to be the best way to monitor the device. In fact, as it will be explained in section 4.5.1, the honeypots are single devices inside a cloud environment, public or private. Therefore, it is less easy to externally monitor the devices, even if it remains possible. Generally, out-of-band monitoring makes sense to monitor networks and not single devices. Configuring such tools seemed overkill in regard to the project's scope.

4.4.3 Event Log

Now that the tools are up and running, valuable information should be defined. Actually, in this current configuration, Cisco Secure Endpoint already collect a lot of data regarding the state of the device. All of these are visible from the Secure Endpoint console, as shown for instance in FIGURE 4.4.1 for a short period of time.

In the use case presented here, it seems straightforward that what needs to be detected is any activity using the remote desktop protocol. Furthermore, the created device will not be used for any other purpose than collecting metrics coming from RDP connections. Also, Windows Firewall has been configured to only accept connections of such type.

[3] For more information about the full Secure Endpoint connector deployment process, see this website.
[4] For more information about the full Orbital deployment process, see this website.

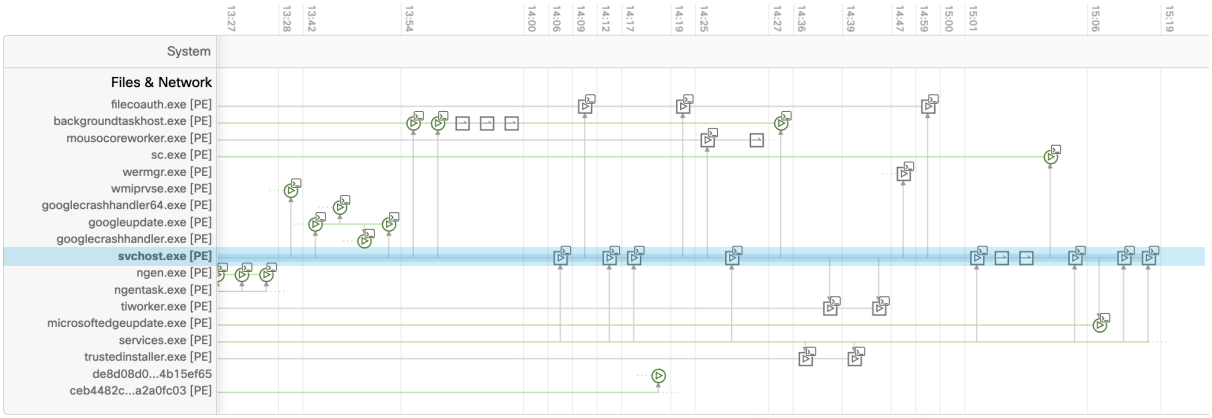


FIGURE 4.4.1
Example of Device Trajectory view for data collected by Secure Endpoint. In blue is highlighted the service host (`svchost.exe`) process, namely used for user authentication via an RDP connection.

Therefore, excepting the Windows classical activity (implying update processes, network connectivity protocols, etc.), only illegitimate RDP activity is expected in the device. Thus, this represents the only interesting load to monitor on the created honeypot.

Secure Endpoint is of course able to detect RDP activity as shown in FIGURE 4.4.1. In this figure, the service host process, ***a.k.a.*** `svchost.exe`, used during a RDP authentication is highlighted in blue. However, even if Secure Endpoint detects a RDP connection, it will not flag it as malicious since RDP is actually a legitimate Windows service. Therefore, even if in the scope of our honeypot, any connectivity using RDP is illegitimate, Secure Endpoint has no way of knowing it. Of course, any file can be declared as malicious from the outbreak control tab of Secure Endpoint administration control. However, any new detected usage of this file will trigger Secure Endpoint making the tool flagging the device using executable as infected and maybe even isolating it. Regarding the case presented here, it is probably not the best option.

A better approach to detect and properly react to remote desktop connection attempts is to use SecureX workflows and Cisco Orbital. As explained in section 2.5.1, SecureX allows to create fully customizable security workflows executed on target devices. The workflow implemented to detect an RDP connection is depicted in FIGURE 4.4.2. More details about the sub-workflows involved in this global diagram are given in FIGURE 4.4.3 and FIGURE 4.4.4. These representations are as close as possible to what their implementation should look like in SecureX's GUI. These workflows as well as all the other ones presented in this writing are publicly available on the GitHub of this thesis.[78]

The monitoring workflow (***cf.*** FIGURE 4.4.2) starts by getting the list of processes running on the honeypot and using the RDP standard port. This actually represents a workflow in itself (***cf.*** FIGURE 4.4.3) first starting by generating an access token for Orbital using Threat Response. This is actually a Cisco standard workflow that will allow SecureX to authenticate to Orbital and perform some queries. Once the token is obtained, the second step executed is to query the honeypot using Orbital to get the list of processes of interest. The query executed by this action is given in CODE 4.3. The very last step of this workflow is to parse the Orbital query result, which is in JSON format.

```
SELECT p.name, pos.protocol, pos.local_address, pos.remote_address, pos.
    local_port, pos.remote_port
FROM process_open_sockets pos LEFT JOIN processes p ON p.pid=pos.pid
WHERE pos.local_port=3389 AND pos.remote_address NOT IN ("", "0.0.0.0",
    "127.0.0.1", "::", "::1","0");
```

CODE 4.3

Orbital query getting the list of processes using the RDP standard network port 3389, the standard number of the protocol used, the local and remote IP address of the hosts involved in the connection as well as the ports they use. Of course this excludes connections coming from the host itself.

Based on the collected information, any network connection using RDP can now be detected. If there is no result for the executed query, the workflow can end. Otherwise, for all the connections in the result, information should be parsed and sent using Webex to the administrator of the honeypot following the workflow given in FIGURE 4.4.4. In the meantime, a forensics snapshot of the device is taken using Orbital in order to detect

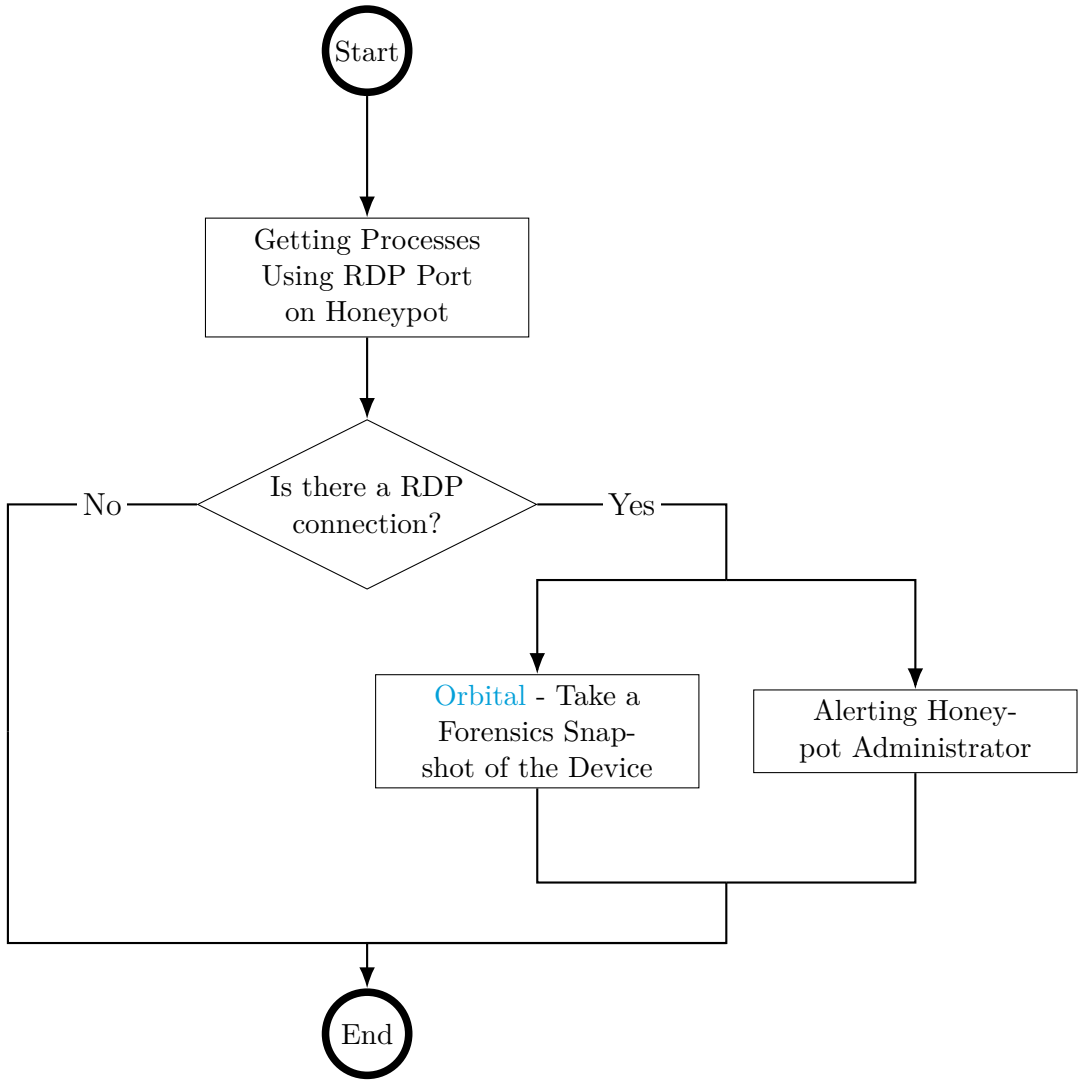


FIGURE 4.4.2

Global monitoring workflow for the created remote desktop protocol honeypot. The Getting Processes Using RDP Port on Honeypot and Alerting Honeypot Administrator are sub-workflows depicted respectively in FIGURE 4.4.3 and FIGURE 4.4.4. In blue are highlighted the atomic operations calling one of the SecureX tool.

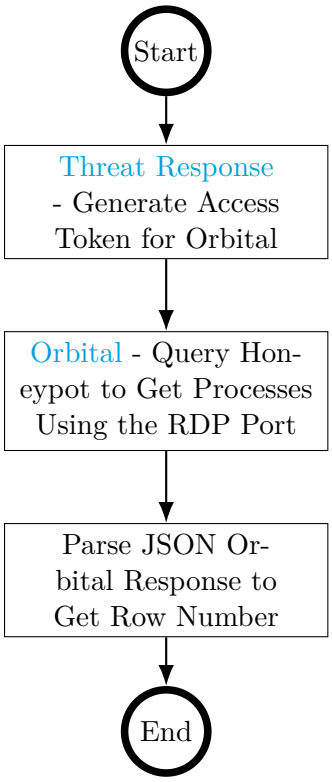


FIGURE 4.4.3

Typical Orbital query workflow used to query the RDP honeypot in order to get the list of process running on RDP port (3389). In blue are highlighted the atomic operations calling one of the SecureX tool.

any change compared to the initial state of the honeypot. Once every RDP connection has been logged using Webex, the workflow can end successfully.

All the SecureX workflows presented in this thesis are triggered based on a time event. This means that every x period of time, where x is configurable, the Orchestration workflow will be launched by SecureX to execute its actions. For the RDP honeypot presented in this chapter, each workflow has been triggered every five minutes, which has been measured as the time for which an instantaneous connection remains in average in Orbital's `process_open_sockets` table.

At first, the isolation feature of Secure Endpoint was also used in the framework in order to disconnect the honeypot from the Internet while experiencing a connection from a foreign host. However, it has quickly been understood that even if this reduced the risk associated to the deployment, it would also have drastically reduced the amount of data collected by the honeypot. Furthermore, regarding the configuration presented in section 4.3, the risk of corruption for the device was already almost nonexistent. For these reasons, putting the honeypot in isolation using Secure Endpoint has been abandoned while defining the monitoring strategy.

By using this complete SecureX workflow, RDP connections can therefore be detected easily and efficiently without declaring any legitimate service as being malicious which looks like a better practice.

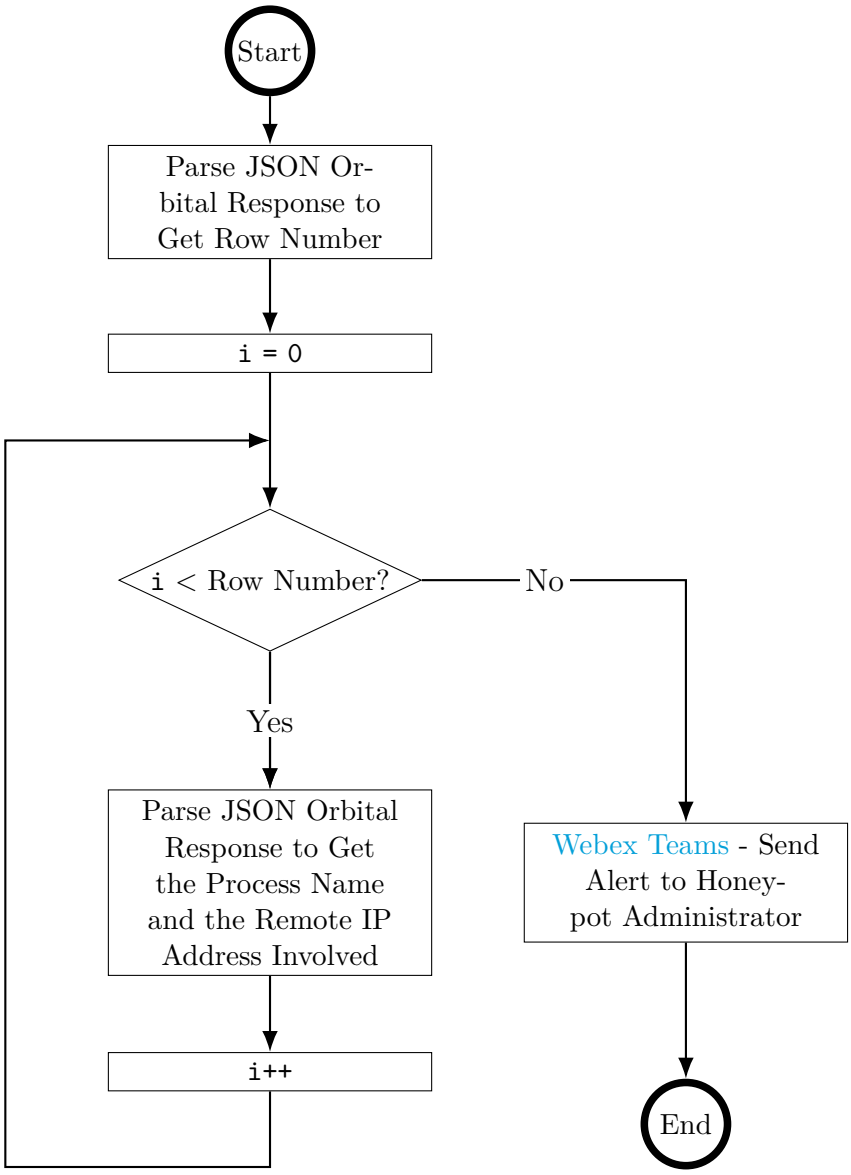


FIGURE 4.4.4

Alerting workflow using Webex to send a message to the honeypot administrator. This workflow needs a JSON Orbital response as an input to parse the results of the Orbital query and get information about the connection(s) to the RDP honeypot it will alert. In blue are highlighted the atomic operations calling one of the SecureX tool.

4.4.4 Critical Event Alerting

As explained in section 2.3.4, when dealing with honeypots, every event or change happening on such a device should be considered as critical (except for normal OS activity as usual). Therefore, in this monitoring strategy, each connection triggers an alert sent to the honeypot administrator using Webex. A typical alert sent using the monitoring workflow presented in this section is given in FIGURE 4.4.5.

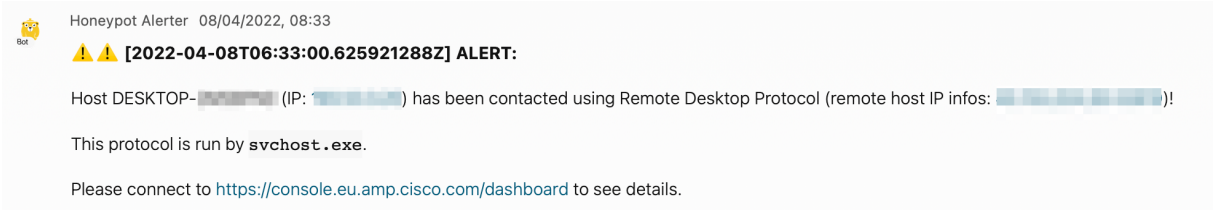


FIGURE 4.4.5
Alert sent using the RDP monitoring workflow to the honeypot administrator.

4.5 RDP Honeypot Classification

First of all, it should seem straightforward that the honeypot defined in this section can be seen as a honey service, impersonating the interaction of the remote desktop protocol while not providing the whole functionalities of a computing environment.

4.5.1 Purpose

To classify the created honeypot based on its purpose, more details regarding its placement should be introduced. Actually this device has not been placed in one, but in two places.

The first one is the Cisco laboratory from the Diegem office. A diagram of the infrastructure of this lab is given in FIGURE A.1.1 of appendix A of this thesis. Same as for all laboratories, this environment is supposed to be aseptic and therefore no illegitimate traffic should be observed within this infrastructure. The honeypot created in this section is a simple and efficient way to ensure that no discovery attacks will take place inside the lab. Indeed, if for instance an attacker launches a port scanning attack for all the devices in the local area network of the lab, this will be detected using the workflow explained in section 4.4. Therefore, inside this infrastructure, the device is definitely part of the detection honeypot family.

The second infrastructure in which the created RDP honeypot has been deployed is part of the University of Liège infrastructure, detailed in FIGURE A.2.1 also from appendix A. This time, the area of deployment is in the public Internet, therefore the honeypot is not part of an internal network anymore. Using a public area for such device intends to collect metrics regarding how RDP attacks work nowadays. Therefore for this deployment, it is clear that the device belongs to the research honeypot family.

4.5.2 Interactivity

Once again, the interactivity level expected for the created honeypot is straightforward since it only emulates a single service being RDP. Therefore, this device can be defined as a low interaction honeypot.

4.6 Conclusion

In this chapter a honeypot able to attract illegitimate traffic detected using SecureX while remaining as simple as possible has been defined. This fits perfectly with the low interactivity honeypot definition which is indeed the family to which the created device

belongs. From the placement of the device, the reader can see that the data collected will not have the same purpose and thus neither the honeypot, as defined in section 1.2 of this thesis.

The monitoring of this device could have been implemented using other tools, which could have given the same result in terms of L&M. However, the unique XDR approach of SecureX greatly eases the monitoring strategy implementation, which has been demonstrated in this chapter. This is namely achieved by offering tools for several different surveillance purposes and interconnect them very efficiently in a centralized application.

This chapter follows the theoretical framework defined in all chapters from part I of this thesis. Using a practical deployment, the honeypot theory has been exemplified for all the pieced put together and not just for some particular steps.

Chapter 6 from part III will show that even if it is simple, the honeypot created in this chapter was able to collect valuable data in a short amount of time.

Simulating an Elasticsearch Deployment in a Public Infrastructure

In this chapter is presented a second honeypot deployed as part of this master thesis. This time, the honeypot intends to help a real partner company to better understand one of the applications it uses daily, the Elastic stack. The motivations behind this deployment will first be introduced, then Elastic's services, the technical details of the deployment and its monitoring strategy will be presented. Finally, the honeypot will be fully categorized following the theoretical concepts presented in part I of this writing.

5.1 Motivations

To understand the emergence of the idea behind the deployment presented in this chapter, it is important to explain the context surrounding it. When starting to think about a new application for another use case, one honeypot presented in chapter 4 was already up and running, the one placed inside Cisco's laboratory at Diegem. With this fully functional proof of concept showing how SecureX could help monitoring such a device, it was time to find some real life problems to solve by using a honeypot, in order to demonstrate the value of such security systems.

The discussion naturally started with the University of Liège and in particular with the *Service Général d'Informatique* (***a.k.a.*** SEGI) to find out a honeypot application that could help daily. As the reader may have understood at this point, there is no limit to what could be implemented inside a honey architecture. However, defining its goal as well as the objectives is primordial not to waste time.

When deciding to use an application, companies usually make some risk and impact analysis in order to make sure that the application follows the internal operational but also security requirements. However, risks are often evaluated based on information published by the vendor of the solution. To study this last aspect in real life environment and thus collect real metrics about an application behavior and not marketing ones, honeypots are perfect candidates. Following this mindset, the research was quickly oriented towards an application used by the SEGI but for which the team did not really have control over its implementation, neither real life metrics about its functioning. Studying such a service using honeypots would therefore allow the SEGI to ensure its performance in terms of security, to have an estimate of the attractiveness of the black hat community for it and maybe even to discover its potential flaws. The first goal of this deployment would therefore be to help the SEGI to have a better visibility on one of the application they

use daily.

After several discussions about the university infrastructure and the tools used day after day by the SEGI to manage it, one proposal for the creation of a honeypot was to simulate the Elastic stack (***a.k.a.*** ELK stack, being an acronym used to describe a stack made of three popular projects: Elasticsearch, Logstash, and Kibana). The ELK stack is used to aggregate, analyze and create visualizations of logs coming from a network. The open source version of the ELK stack is deployed on-premise inside the university of Liège's infrastructure. FIGURE A.3.1 from appendix A gives a global overview of the current deployment of the ELK stack managed by the SEGI. This tool collect every day several thousand of gigabytes of critical data regarding the university infrastructure which should make it attractive for any insider. Furthermore, even if the ELK stack is a well known tool widely spread in the security community, the SEGI had no real clue about the security limitations of this applications.

For all the points discussed supra, it has been decided to create a honeypot simulating an ELK stack deployment, placed in the public internet space of the university of Liège specially created for this master thesis (***cf.*** FIGURE A.2.1). The goal of this deployment is therefore to measure the enthusiasm of the black hat community for this type of applications as well as to potentially discover some flaws inside its open source implementation.

5.2 Elastic Stack

The Elastic stack, popularly known as ELK stack, is an open source software stack yet maintained by Elastic. In 2015, around three years after the creation of the Elastic NV company[79], the application Beats joined the stack, turning it into a four layer project. The different levels of this solution as well as their interactions are depicted in FIGURE 5.2.1.

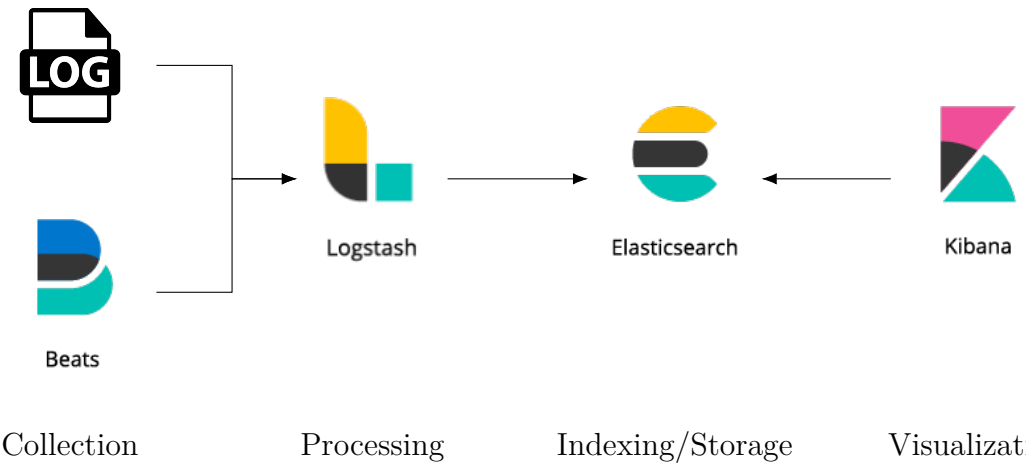


FIGURE 5.2.1
Global overview of the classical interaction between all the components of the ELK stack. Typically, data are aggregated into log files, by using Beats or come directly from the network and then are processed by Logstash. After what, they are stored into the Elasticsearch search and analytics engine. Finally, Kibana can be used as a user friendly front-end to visualize the stored data.

As already briefly mentioned in section 2.3.3, one of the numerous ELK stack goals

is to try to address the operational management problem coming from unusable amount of noisy and not standardized logs. The solution proposed provides a whole suite of applications allowing to collect, process, store and visualize easily information coming from a whole infrastructure. The following sections shortly present each components of the ELK stack. These sections serve only to introduce the key concepts behind Elastic's layers. For more detailed information about this technology, the reader is invited to consult the article *The Complete Guide to the ELK Stack* wrote by D. Horovits[81] as well as Elastic official documentation.[82]

5.2.1 Elasticsearch

As already mentioned before, Elasticsearch is used to store data transiting in the stack. Whether these last are structured, unstructured, text, numerical,... Elasticsearch will index and store them in order to offer fast search functionalities.

Elasticsearch is a distributed search and analytics engine built on Apache Lucene.[81] It is the central element of the ELK Stack used primarily for search and log analysis, but becoming today one of the most popular database systems.[80] In term of architecture, this distributed application is categorized as a NoSQL database and obeys the representational state transfer (REST) constraints, ***i.e.*** said RESTful.

One of Elasticsearch's most interesting feature is its distributed nature. While the data and query volume grows, the deployment can be adapted easily to support the load thanks to this design. Furthermore, this application has been implemented to hide the whole complexity usually brought about by the distributed systems. This aspect of Elasticsearch is in fact largely transparent.[83]

5.2.2 Logstash

As explained in section 2.3.3, when it comes to log analysis, two main difficulties arose being the lack of standardization regarding log structure as well as the among of noisy irrelevant logs generated by infrastructures assets. Logstash is tasked with the crucial labor of parsing data contained inside those anarchic log files and eventually of their restructuring.

Logstash define a three-stage pipeline implemented in Java and Ruby.[84] The different stages of the pipeline are first the input, reading raw data from several streams (***e.g.*** files, Syslog, Beats, ...). Then comes the second stage being the filtering, based on regular expressions. This stage represents the actual processing of events taking place. After these, the last stage in Logstash's pipeline, being the output, can be executed which allows to restructure the previously matched events and send them to their output destinations.

5.2.3 Kibana

Kibana is the front-end application of the ELK stack, helping search, analyze and visualization of the data indexed in Elasticsearch. This application also implements a GUI allowing to monitor the ELK deployment(s) to which it is attached.

Technically speaking, Kibana is a classical browser based front-end for Elasticsearch mostly developed in TypeScript.[84]

5.2.4 Beats

Beats is made of a collection of open source log shippers used to collect data from devices and send them either to Logstash for formatting or directly to Elasticsearch for indexing and storage.

The architecture of Beats is made of several packages (***a.k.a.*** Beat), ***e.g.*** Filebeat for forwarding logs coming from files or Metricbeat to collect metrics from systems and services called the officially supported Beats in Elastic's GitHub.[84] This application uses the Go `libbeat` library, implementing the Application Programming Interface (API) used by all Beats to ship data.

5.3 Creating an Elastic Stack Honeypot

As explained in section 5.1 the goal of this deployment is to provide better control over the Elastic stack to a company. Therefore, to be efficient, most of the technical decisions made in this chapter have been taken to remain as close as possible from the production environment of this firm. Some of them have also been taken in order to increase the credibility of the created honeypot, they will be highlighted further down. The deployment used by this corporate is given as reference in appendix A, in FIGURE A.3.1.

Even though the honeypot created in this section should remain close to the one in production it tries to simulate, it does not mean that it cannot be simplified. Indeed, technical decisions such as the operating system used for the hosts running the ELK applications, the version of these lasts or the type of informations they will treat among many others, are important and should be considered for the purpose of this deployment. However, the among of data treated, the number of nodes used, etc. can be reduced in order to not waste resources without reason. Oxford dictionary defines a sample as

> *A small amount of a substance taken from a larger amount and tested in order to obtain information about the substance.*

This is exactly what this honeypot will be, a smaller representation of the considered infrastructure tested to study the behavior of production one.

To remain close of the studied infrastructure, each virtual machines in the deployed honeypot will use Rocky Linux 8.5 as operating system. The deployed Elastic stack layers are the ones from version 8. However, simulating a gigantic infrastructure, made of thousands of assets from which data are collected as well as creating log aggregators services to get all of these information seemed completely overkill regarding the goal of the deployment. This is why first, a small containerized application has been created in order to simulate classical infrastructure log generations and feed the ELK stack implementation. Also, the number of elements composing the stack has been kept smaller than in the production environment it simulates while not being minimal. Indeed, the whole ELK stack can in fact run in a single device while here several are created as it will be explained in section 5.3.1. Keeping all the services in a single machine would probably have greatly eased the deployment, however it has been thought that studying the network behavior of the Elastic elements was more important than oversimplifying the honeypot.

To add credibility to the deployment, a usage scenario has been imagined for the generated honeypot. Indeed, not everybody use the ELK stack and, to make the deployment appears as legitimate, a small effort justifying it seemed adequate here. Regarding the purpose of the ELK stack described in section 5.2, it seems obvious that this tool is mostly used by companies and not ordinary people. This is why Typhoon[1] has been imagined from scratch, as being a small university spin-off company using the university Internet infrastructure. In the following paragraph, the reader will notice that Typhoon appears many time in the configuration, this paragraph explain its presences. More information about the nature of this newly designed element will be given in section 5.5.3.1.

5.3.1 Elastic Stack Structure

The deployed ELK infrastructure is given in FIGURE 5.3.1. This infrastructure is one of the most basic configuration of the Elastic stack made of two Elasticsearch nodes storing data coming from a Logstash instance and using a Kibana server to visualize them. The Log generator application feeding Logstash will be developed in section 5.3.2.

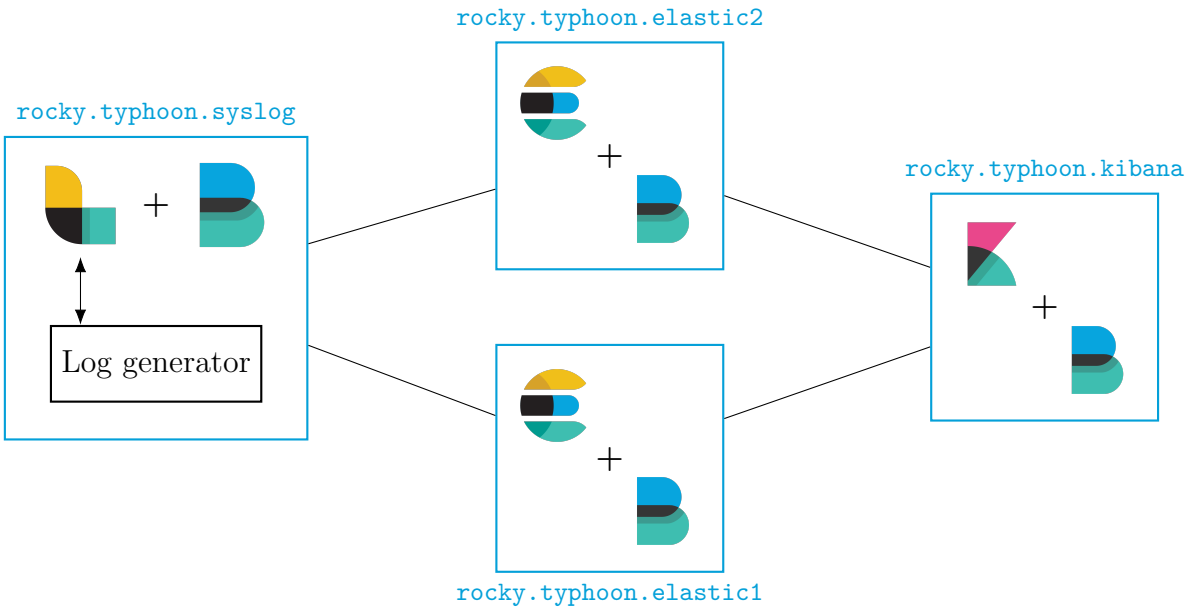


FIGURE 5.3.1

Infrastructure of the honeypot simulating an Elastic stack deployment. This honey deployment is made of four different machines (represented by blue rectangles) using Rocky Linux 8.5 as OS. One runs Logstash, fed by a containerized application simulating an infrastructure log aggregator behavior (***a.k.a.*** Log Generator, ***cf.*** section 5.3.2). Two machines are the Elasticsearch nodes, representing the Elasticsearch cluster. The last device is dedicated to the Kibana application. Each of the ELK services is monitored using Beats, which has also been illustrated in the diagram. The production host names following the convention `os_used.fake_company_name.usage` are depicted in blue around their machine representations.

The configuration is made of not one but two Elasticsearch nodes, the second one existing for redundancy purpose. While simulating a real life deployment, one needs to keep in mind the common practices of the industry. Even if in this casus, data losses or performances are not of the utmost importance, the deployment needs to appear legiti-

[1] The most insightful will have noticed that this word is actually a non-complete anagram of the word honeypot (only missing an "e").

mate to follow the honeypot definition introduced in section 1.1. Redundancy is usually implemented for critical assets and the ELK stack would definitely be part of these.

In the following sections are explained the configuration of the services from the infrastructure described above. For the Elastic stack elements, the installation subtleties are not covered in this thesis since they are OS dependent and usually straightforward. All the configurations files discussed in these sections are available on the GitHub repository of this thesis, in the `config` folder. Part of these files might also be quoted below in order to highlight them.

5.3.2 Simulating Infrastructure Log

Usually, Elastic stack is deployed to monitor big infrastructures, made of a lot of assets generating a lot of event logs to process. However, regarding the goal of the honeypot created in this section, the only thing that matters is to feed the ELK deployment with data, not that this data come from legitimate assets. With this mindset, it has therefore been decided to create an application generating logs resembling those inside real life infrastructures while being in fact completely fake data. The source code of this application, called Log Generator, is available on GitHub[85] and its general architecture is given in FIGURE 5.3.2.

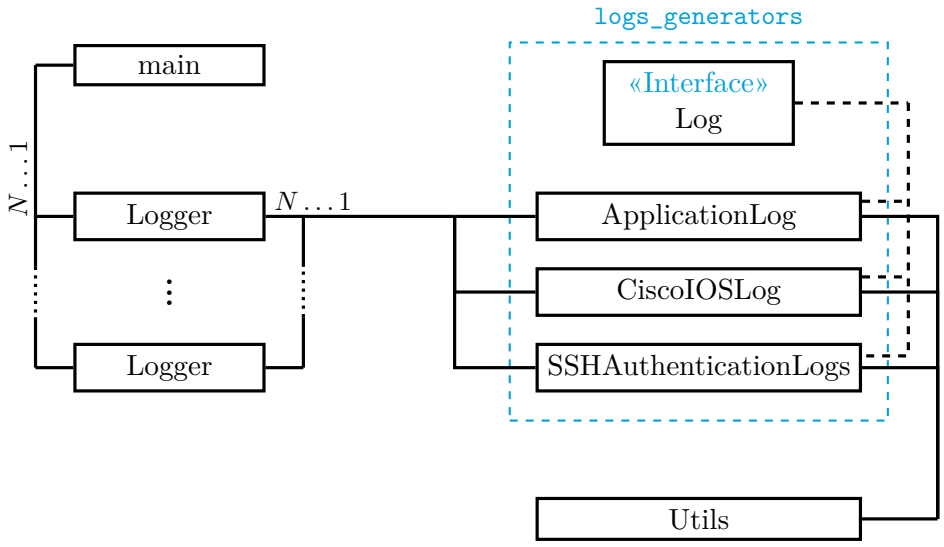


FIGURE 5.3.2

Diagram of the implementation of the Log Generator application. As usual, `main` is the program's entry point which creates several `Logger` threads. Each `Logger` generates `Log` of different types, each one having its own implementation under the `log_generators` package (framed by the dashed blue line) inheriting from their `Log` parent interface (relation shown by the dashed black lines). The `Utils` class brings together a range of utilities helping in particular to generate the false data appearing in the logs.

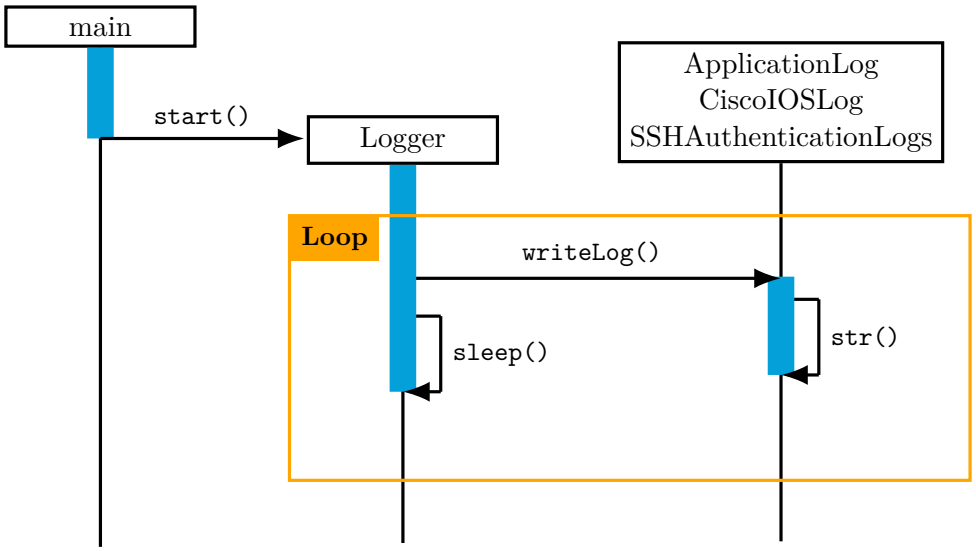


FIGURE 5.3.3

Dynamic diagram of the Log Generator implementation showing its typical excecution. Each `Logger` thread is launched by `main`, this last being only active for this purpose. After what, each `Logger` executes in an infinite loop (represented in orange), alternating between log creation and sleep time. Logs are created by the various implementations from the `log_generators` package (***cf.*** FIGURE 5.3.2).

Log Generator is a simple multi-threaded Python application generating logs in their different standard formats (***e.g.*** Cisco IOS logs, SSH authentication logs, ...) and writing them into an output file. In the scope of this thesis, as already mentioned several time, the only purpose of this application is to generate data fed into the ELK stack and which is looking as legitimate. FIGURE 5.3.3 gives the dynamic diagram of the Log Generator excecution. Basically, main creates each Logger thread of the program and each of these threads executes in an infinite loop, during which it alternates between creating logs in the output file (by using the different log generator from the package `log_generator`) and going to sleep mode. The types of log created and the duration time during which a thread sleep are both random but configurable using the defined constants for the application. The content of the logs (***e.g.*** username, IP addresses, log level, etc.) is also highly random and relies heavily on the Faker[86] Python package.

To ensure portability, resource efficiency and fast deployment, the Log Generator application has been containerized using Docker. The CODE 5.1 and CODE 5.2 give respectively the `Dockerfile` and `docker-compose.yml` files used to create the container running the application. The complete description of these simple files goes beyond the scope of this master thesis. However the line 5 of CODE 5.2 merits some explanations since it is responsible for the creation of the file `launcher.sh` given in CODE 5.3.

While creating a container, with Docker Compose for instance, every action it will execute will have `root` privileges by default in the containerized environment. For instance, if the container creates a file on the device it runs on thanks to a volume binding, reading this file will only be possible from a `root` account. To ensure security and access control, it is good practice to grant a container with the same privileges as the user creating the service, as long as the containerized application does not require higher privileges of course. To do that, the field `user` of the defined services in the `docker-compose.yml` file

can be used, allowing to set a group and user ID.[2] However, depending on the operating system on which the container is created, predefined environment variables containing the group and user ID may have different names or not be available. The `id`[87] utility can be nevertheless used to determine these identifiers on compliant systems. It is what is performed by the script `launcher.sh` from Code 5.3, where customs environment variables are defined then exported in the environment of `docker-compose.yml` file. One can see that the group and user identifier are respectively set to the variables `DOCKER_GID` and `DOCKER_UID` using the `id` command before calling the `docker-compose` one, which creates the container using them to set the `user` field discussed supra.[3]

```
FROM python:3.8-slim
ADD src /src

RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir Faker && \
    mkdir ./output && chmod ugo+rwx ./output

CMD ["python", "-m", "src.main"]
```

CODE 5.1

`Dockerfile` text document containing all the commands to assemble the container used to run the Log Generator application.

```
version: "3.9"
services:
  logger:
    container_name: "logger"
    user: "${DOCKER_UID}:${DOCKER_GID}"
    build: .
    volumes:
      - "./output:/logger/output"
```

CODE 5.2

Docker Compose YAML file used to configure Log Generator's services.

```
#!/bin/sh
export DOCKER_UID=$(id -u)
export DOCKER_GID=$(id -g)
docker-compose up -d
```

CODE 5.3

Script `launcher.sh` used to export the user and group IDs from a device and launch the container running the Log Generator application.

5.3.3 Configuring Elasticsearch

Now that data is being created to fed the elements of the Elastic stack, it is time to configure each of them. As mentioned above, Elasticsearch is at the center of the ELK deployment, therefore it is logical to start by configuring this service.

[2] Notice that setting this the field `user` in the `docker-compose.yml` file is equivalent to use the option `-u` (or `-user`) from `docker` command line interface.

[3] Of course, since this script is a Bash script, even if its existence makes the container creation even more portable, it will require the host calling it to be compatible with this technology. However, this script can also be easily translated into other shell languages.

Since the deployed honey infrastructure works with two Elasticsearch nodes, two devices with this application installed are therefore required. Once the Elasticsearch application and all its dependencies installed on each device, its whole configuration can be done through the file `/etc/elasticsearch/elasticsearch.yml`. These are given for the first and second node respectively in the `elasticsearch1.yml` and `elasticsearch2.yml` files from the `config/TLS_enabled` folder available on the GitHub repository of this thesis.[78] The configuration of these two nodes is quite straightforward and explaining it in details would go beyond the subject of this thesis. Note however that the naming convention is the same as the one used in FIGURE 5.2.1. Nevertheless, the network part of this last as well as consideration regarding security can be briefly reviewed.

5.3.3.1 Network

As mentioned before, the deployment of the different elements composing this honeypot simulating the ELK stack runs over several distinct devices, therefore implying network usage. By default Elasticsearch is only accessible on `localhost` but setting the field `network.host` to a different address allows to expose an Elasticsearch node to the network.

Still regarding network configuration, Elasticsearch uses the first free port found starting from 9200 by default. However, a specific port can also be set using the `http.port` field.[4]

CODE 5.4 shows the configuration part exposing the node to the network, making them accessible from outside the device. This sample is the same for both the Elasticsearch nodes, only the IP addresses used will of course differ and be set to the device hosting the service IP address.

```
# --- Network ---
network.host: "b.b.b.b"
http.port: 9200
```

CODE 5.4

Part of the Elasticsearch configuration file for the considered honeypot deployment used to expose the service on the network. In this YAML file sample, the used IP address has been anonymized. Notice also that this configuration is the same for the two nodes except that their own IP addresses are used of course.

Of course, since ports are used either to contact the two nodes or to allow them to communicate, the traffic they may receive should be accepted by the firewall of the device. Rocky Linux, the OS of the machines presented here, uses Firewalld as default firewall which is not configured to allow Elasticsearch traffic by default. To create the rules forwarding this last, the commands given in listing CODE 5.5 can be used. Of course, these commands consider that Elasticsearch default port has not been changed. Also notice that port 9300 is used by Elasticsearch nodes to communicate between them. Therefore for a single node infrastructure, this port has no need to be opened.

```
sudo firewall-cmd --permanent --add-port=9200/tcp
sudo firewall-cmd --permanent --add-port=9300/tcp
```

[4] Notice that if this default value is changed, every services connecting to Elasticsearch using HTTP API will need to take that into account. For instance, a Logstash application configured to send its data to Elasticsearch configured as well will also need to change the connection port.

```
sudo firewall-cmd --reload
```

CODE 5.5
Firewalld commands allowing Elasticsearch traffic.

5.3.3.2 Security

As mentioned in section 5.3.1, keeping a honeypot deployment simple is one thing, however this deployment needs to remain credible in order to be effective. As will be explained in section 5.5.1, the honeypot described in this chapter is deployed inside a public infrastructure. Even if deploying the ELK stack in such an environment does not seem too wild, betting on the fact that a lot of companies will not secure their production log infrastructures exposed in the Internet appears way less reasonable. Indeed, in the 21st century, using HTTP for sensitive communications such as the ones supposed to be managed by the Elastic stack should no longer be relevant. It is in this perspective (***i.e.*** in order to add credibility to the honeypot) that encryption using SSL/TLS has been configured for all Elastic services described in this section.

```
# --- BEGIN SECURITY AUTO CONFIGURATION ---
# Enable security features
xpack.security.enabled: true
xpack.security.transport.ssl:
  enabled: true
  verification_mode: certificate
  key: certs/nodeX.key
  certificate: certs/nodeX.crt
  certificate_authorities: certs/ca.crt

# Enable encryption for HTTP API client connections, such as Kibana,
   Logstash, and Agents
xpack.security.http.ssl:
  enabled: true
  verification_mode: certificate
  key: certs/nodeX.key
  certificate: certs/nodeX.crt
  certificate_authorities: certs/ca.crt
```

CODE 5.6
Part of the Elasticsearch configuration file for the considered honeypot deployment used to configure security capabilities for Elasticsearch nodes communications. In this YAML file sample, the Elasticsearch nodes, certificates names as well as IP addresses used have been anonymized. Notice also that this configuration is the same for the two nodes, except that the names design their respective data of course.

From Code 5.6, the reader can see that both traffics for basic transport (***i.e.*** connections between ELK nodes) or HTTP have been protected by encryption using Secure Sockets Layer (SSL) technology. SSL relies on the usage of certificates to encrypt communication data. These last has been generated using the `elasticsearch-certutil` utility coming with Elasticsearch application. Elasticsearch not being one of the trusted root authorities able to issue certificates and sign them, this utility should not be invoked to generate certificates used to secure production data. However, regarding the fact that the data secured here are in reality fake ones traveling inside a honey infrastructure, creating the used certificates for the whole ELK configuration with `elasticsearch-certutil` was judged sufficient.

5.3.4 Configuring Logstash

Like for Elasticsearch nodes, the Logstash ones can also be fully configured using a YAML files findable under `/etc/logstash/logstash.yml`, visible in its final configuration state in the `logstash.yml` file from the `config/TLS_enabled` folder available on the GitHub repository of this thesis.[78] This application does not require a a lot of configuration. Indeed, Logstash only ships data to a destination without needing to be spontaneously contacted, therefore no network exposure of the service is required. Following the security considerations made above, the shipped traffic by Logstash has however also been protected and this will be reviewed in section 5.3.4.1.

The most important configuration part of Logstash not lies in the application configuration itself, but in the pipeline one. Logstas pipeline's stages are defined into `.conf` files loaded by the application at startup. By default, Logstash will load all the `.conf` files located in the path `/etc/logstash/conf.d/`, which has been used in this deployment. The pipeline configuration for this particular use case will be reviewed below in section 5.3.4.2.

5.3.4.1 Security

Since security has been activated on the Elasticsearch side of the deployment, Logstash should align and secure its communications with the Elasticsearch nodes too. Code 5.7 shows the parameters used to secure this traffic.

```
# ----------- X-Pack Settings (not applicable for OSS build) -----------
#
# X-Pack Monitoring
# https://www.elastic.co/guide/en/logstash/current/monitoring-logstash.
    html
xpack.monitoring.enabled: true
xpack.monitoring.elasticsearch:
  hosts: ["https://b.b.b.b:9200", "https://c.c.c.c:9200"]
  username: "logstash_writer"
  password: "PASSWORD HERE"
  ssl.certificate_authority: "/etc/logstash/certs/ca.crt"
```

CODE 5.7

Part of the Logstash configuration file for the considered honeypot deployment used to configure security capabilities for the node communications. Security is achieved by credentials authentication (instead of certificate infrastructure, as presented in section 5.3.3). In this YAML file sample, the Logstash password as well as the Elasticsearch node IP addresses used have been anonymized.

One can see that this time, the security relies on a user/password authentication mechanism and uses a certificate authority (CA) to validate the identification. Theses credentials can be easily configured inside the Elasticsearch nodes using the `elasticsearch-setup-passwords` utility. Of course, since this utility set sensitive data inside the Elasticsearch database, it will be executed on the device hosting this service. Note that this tool allows to setup password for Elasticsearch built-in users, from which can be find `logstash_system` used in the configuration presented in 5.7.[5]

[5]For more information about Elastic stack built-in users, please refer to this website

5.3.4.2 Pipeline

As explained in section 5.2.2, Logstash defines an information treatment pipeline and this one is fully configurable using Logstash's `.conf` files. The `.conf` file used in the scope of this Logstash configuration is given in Code 5.8 and also available on the GitHub repository of this thesis.[78] The following paragraphs will review each of the three pipeline stages that it defines.

```
input {
  file {
    path => "/path/to/log-generator/output/infrastructure_logs.log"
    start_position => "beginning"
  }
}

filter {
  grok {
    match => { "message" => "%{TIMESTAMP_ISO8601:time} %{LOGLEVEL:logLevel} \[%{GREEDYDATA:app}\]:%{GREEDYDATA:logMessage}" }
    match => { "message" => ['\*%{SYSLOGTIMESTAMP:syslog_timestamp} %{GREEDYDATA:facility}-%{INT:severity_level}-%{GREEDYDATA:facility_mnemonic}: %{GREEDYDATA:description}'] }
    match => { "message" => ['%{SYSLOGTIMESTAMP:syslog_timestamp} %{HOSTNAME:hostname} sshd\[%{INT:pid_sshd}\]: %{WORD:connexion_attempt_status} (user|password for)( invalid user)? %{USERNAME:username} from %{IPV4:sshd_remote_connection_ip}( port %{INT:sshd_remote_connection_port} %{GREEDYDATA})?$'] }

    add_tag => [ "matched" ]
  }
}

output {
  if "_grokparsefailure" in [tags] {
    file {
      path => "/home/admin/Desktop/fail-%{type}-%{+YYYY.MM.dd}.log"
    }
  }
  if "matched" in [tags] {
    elasticsearch {
      hosts => ["https://b.b.b.b:9200", "https://c.c.c.c:9200"]
      index => "logstash-%{+YYYY.MM.dd}"
      user => "logstash_writer"
      password => "PASSWORD HERE"
      cacert => '/etc/logstash/certs/ca.crt'
    }
  }
}
```

CODE 5.8

Logstash's pipeline configuration file for the deployed ELK stack. This `.conf` file is loaded automatically from `/etc/logstash/conf.d/` at the application startup and contains the definition of the input, filter and output stage of Logstash's pipeline.

Input

The input stage of the pipeline is configured to get data coming from a file located in `/path/to/log-generator/output/infrastructure_logs.log`. As the reader would

have probably figured out, the file is the one generated and fed of logs by the Log Generator application presented in section 5.3.2. Of course, several other inputs can be added as data sources for the pipeline, being of different types, as long as they are part of the supported input plugins which list is given in the Elastic documentation.[82]

When installed on a device, by default Logstash ran using a user of the same name with limited access rights on the machine. This application read files that might be protected and therefore higher privileges might be required by Logstash in order to access them. In the production environment mimicked by the deployment described in this chapter, each Logstash process is ran with `root` privileges on each device, not limiting its access right to any stream anymore. To avoid any permissions issues as well as to stay once again as close as possible as the simulated deployment, Logstash has therefore been granted with super user privilege also for this deployment. To configure this, one can simply add the `logstash` user (being the default one used by Logstash) in the `root` group on the device by using the command given in Code 5.9.

```
1 sudo usermod -a -G root logstash
```

CODE 5.9
Command allowing to add the Logstash service default user `logstash` to the `root` group on a device.

Filter

The second stage of the pipeline has been configured to filter the logs fed to Logstash using Grok regular expressions (regex). Grok is an high level regular expression definition tool based on Oniguruma library. A lot of regex matching well known log structures are already defined in Grok syntax (***e.g.*** `TIMESTAMP_ISO8601` or `LOGLEVEL` from Code 5.8). However, there is no limit of the regex definition. Any user can create custom ones using the Oniguruma syntax and embed it to her or his Grok's matching rules.

Of course, the regular expressions match every data created by the Log Generator application inside Logstash input file. To ensure this, functional testing has been performed using the exact Grok regex as the ones given in Code 5.8.

The reader can also notice that a tag named `matched` is added to each filtered data (***i.e.*** log matched by one of the Grok regex) at the filter pipeline's stage. This tag is useful for debugging purpose as it will be explained in the next paragraph.

Output

The final stage of the pipeline have, as for the two previous ones, quite a straightforward configuration. One can directly understand that the two anonymized IP addresses from line 26 of Code 5.8 are in fact the locations of the two Elasticsearch nodes of the infrastructure to which data marked with the `matched` flag are directly send. Elasticsearch group data using indexes, which ones are defined daily following the `logstash-%+YYYY.MM.dd` naming convention, where `%+YYYY.MM.dd` is the ISO 8601[88] date format defined using Joda syntax.[89]

If the tag `matched` is not attached to the event reaching the output stage, then it is marked with `_grokparsefailure`. This is the default tag indicating that no rule has been able to match the input event during the filter stage. Since each log generated in Logstash's input file by the Log Generator should be matched and indexed in Elasticsearch, if one event is marked with the `_grokparsefailure` tag, it therefore means that

some bug exists either in the Log Generator implementation or in the Grok matching rules. To ensure easy troubleshooting, the potentially unmatched logs are redirected directly to a file placed on the desktop of the device running Logstash. If such file is created, it will contain information about unmatched events that could be easily observable and allow the fast correction of the implementation.

5.3.5 Configuring Kibana

Like for the previous nodes of the created stack, Kibana can be entirely configured by using a YAML file being `/etc/kibana/kibana.yml`. This file in its final form for the deployment considered in this chapter is given in the `kibana.yml` file from the `config/TLS_enabled` folder available on the GitHub repository of this thesis.[78] Kibana has probably the most straightforward of the global configurations considered in this section. However, again in order to increase the credibility of the honeypot, some small tweaks have been made to its default configuration regarding both network and, as for each nodes, security. The following subsections present the mindset adopted behind each of these small configurations as well as the tweaks themselves.

5.3.5.1 Network

Code 5.10 presents the fields used to configure the network behavior of the Kibana service.

```
# --- Network ---
server.host: "d.d.d.d"
server.port: 443
server.publicBaseUrl: "https://kibana.typhoon.uliege.be"
```

CODE 5.10

Part of the Kibana configuration file for the considered honeypot deployment used to expose the service on the network. In this YAML file sample, the used IP address has been anonymized. Notice also that the service port has been changed to use HTTPS default one (443) instead of the one used usually by Kibana services (5601). Finally, since an URL has been register for the website inside ULiège's DNS, it is also assign to the service.

In this code sample, one can see that first the `server.host` has been configured to the IP address of the host running Kibana's process. The default value of this field is `localhost`, which does not expose the Kibana service to the network as for the Elasticsearch nodes. Of course in the use case of this honeypot deployment, Kibana needs to be accessible from the outside world to be efficient.

If the first field is the IP address of the host running Kibana, the second cannot be other than the network port on which this service will listen for connection. The default port of Kibana is 5601 but here it has been overwritten to the port 443, which is the default port used for HTTPS services. Behind this configuration, a small assumption has been made being the fact that services running on the most famous protocols or application standard ports would be more enticing then less famous ones. Remember that since the Kibana instance configured here is part of a honeypot, every possible method attracting insiders should be considered. Chapter 7 from III will discuss if making this assumption was reasonable or not. Of course, since port 443 is used and Kibana service has been configured to be accessible from outside the device where it runs, one should not forget to create a firewall rule allowing traffic to reach this application as usual. To do this, since

the device hosting Kibana is once again a Rocky Linux one running Firewalld, the simple code lines given in Code 5.11 will suffice to allow HTTPS reaching the host.

```
sudo firewall-cmd --permanent --add-port=443/tcp
sudo firewall-cmd --reload
```

CODE 5.11
Firewalld commands allowing Kibana traffic.

Last but not least, the final field relative Kibana's network configuration is its public base Uniform Resource Locator (URL). This field specifies the public URL at which the created Kibana instance will be available for users. As the reader may have noticed, the web address `https://kibana.typhoon.uliege.be` is suffixed by the university of Liège domain. This is due to the fact that this URL has been created into ULiège's DNS server. As explained in section 3.3.1, this represents a honeytoken but section 5.5.3.2 will discuss it in more details.

5.3.5.2 Security

Still following the goal of increasing the credibility of the created honeypot, once again encryption has been enabled for communication between Kibana and the outside world. Code 5.12 presents the part of Kibana's global configuration achieving this function.

```
server.ssl:
  enabled: true
  certificate: "/etc/kibana/certs/typhoon-kibana.crt"
  key: "/etc/kibana/certs/typhoon-kibana.key"

elasticsearch:
  hosts: ["https://b.b.b.b:9200", "https://c.c.c.c:9200"]
  username: "kibana_system"
  password: "PASSWORD HERE"
  ssl.certificateAuthorities: ["/etc/kibana/certs/ca.crt"]
```

CODE 5.12
Part of the Kibana configuration file for the considered honeypot deployment used to configure security capabilities for the node communications. Security is achieved by credentials authentication for exchange between Kibana and the Elastic node. HTTPS communications with other users are managed using encryption certificates. In this YAML file sample, the Kibana password as well as the Elasticsearch node IP addresses used have been anonymized.

One can see from the piece of code quoted supra that two encryption configurations are required. The first one is relative to HTTP traffic generated between Kibana and users. Indeed, to make Kibana run over HTTPS instead of HTTP and therefore secure communications between an user and this service, a certificate is required. As written in section 5.3.3.2, even if it is not the best thing to do for production environments, the certificates for this ELK stack honeypot have been generated using the `elasticsearch-certutil` utility and this certificate is no exception. Since this tool generates certificates using an unreliable certificate authority (CA)[6] to sign them, browsers will normally automatically block the access to Kibana's home web page, which can be seen as a limitation of the deployment. This will be discussed in more details in chapter 8.

Once HTTPS for users communications protected, the other important traffic to secure

[6]This not trusted certificate authority is Elastic Certificate Tool Autogenerated CA.

is the one between Kibana and the Elasticsearch nodes. Except for the variable name, the reader can see from Code 5.12 that this is implemented very similarly than for the Logstash configuration, ***i.e.*** by using a username/password pair for authentication to the nodes as well as a CA certificate to sign it and therefore ensure trust.

Once this configuration achieved, Kibana can be accessed from any web browsers or via its API. Any of its network traffic will be encrypted, which should therefore make the service looks presumably legitimate for the outside world.

5.3.6 Configuring Beats

Once every node running the applications forming the acronym ELK is configured, it is always good practice to monitor their behaviors. This is the main purpose of Beats, youngest member of the Elastic stack, as already describe in section 5.2.4. Enabling Beats in each nodes configured in sections 5.3.3-5.3.5 represents the last configuration stage of the simulated services inside the created honeypot. The configuration file of Beats, `/etc/metricbeats/matricbeat.yml` is the exact same for all deployed Beats application and can be seen in one of the files under the `config/TLS_enabled/Beats` folder on the GitHub of this thesis.[78] This file namely defines where to find Elasticsearch node(s) for data indexing as well as the connection mechanism to use to reach them. It also define the Kibana service, allowing Beats to connect to it in order to configure example dashboards, visualizations, and searches to help user to manage the data it generates. Code 5.13 highlight this configuration part of the Beats applications from the Kibana node of the deployment.

```
setup.kibana:
  host: "https://d.d.d.d:443"
  ssl.enabled: true
  username: "elastic"
  password: "PASSWORD HERE"
  ssl:
    verification_mode: "certificate"
    certificate_authorities: ["/etc/elasticsearch/certs/ca.crt"]

output.elasticsearch:
  hosts: ["b.b.b.b:9200", "c.c.c.c:9200"]
  protocol: "https"
  username: "elastic"
  password: "PASSWORD HERE"
  ssl:
    verification_mode: "certificate"
    certificate_authorities: ["/etc/elasticsearch/certs/ca.crt"]
```

CODE 5.13

Beats configuration part describing the Kibana and Elasticsearch nodes with which the application interacts as well as the connection mechanism (HTTPS using CA certificates and username/password authentication).

As shown in Code 5.13, connections to both Elasticsearch nodes as well as the Kibana one are made using HTTPS and username/password authentication, using as usual a CA certificate to ensure trust.

As already stated in section 5.2.4, a collection of Beat (***i.e.*** module) composes the

whole application and each of them can be activated from `metricbeat` command-line interface (CLI).[7] Of course, each of the ELK stack application has its own Beat, implemented specially to monitor its behavior. These Beats modules are obviously configurable, using YAML files loaded by default at the startup of the application in `/etc/metricbeat/module.d/`, which has been used in this configuration. For each of the honey deployment node where Beats has been installed, these files are given in the `/config/TLS_enabled/Beats` folder on the GitHub of this thesis.[78] Since these module configuration files are very similar from one another, only one of them is reviewed in this section and is given in Code 5.14.

```
1  # Module: kibana
2  # Docs: https://www.elastic.co/guide/en/beats/metricbeat/8.2/metricbeat-
       module-kibana.html
3
4  - module: kibana
5    xpack.enabled: true
6    period: 10s
7    hosts: ["https://d.d.d.d:443"]
8    username: "elastic"
9    password: "PASSWORD HERE"
10   ssl:
11     verification_mode: "certificate"
12     certificate_authorities: ["/etc/elasticsearch/certs/ca.crt"]
```

CODE 5.14
Beats module configuration file `kibana-xpack.yml` enabling the monitoring of the Kibana node of the deployment.

The configuration from Code 5.14 defines the module named `kibana` used to monitor Kibana instances. To allow this, it gives the URL(s) of Kibana's instance(s) to monitor as well as the connection credentials for it/them and the connection mechanism used. Most of the module configuration files look like the one quoted supra, simply describing the access to the instances Beats needs to monitor using these modules.

5.3.7 Verifying the Configuration

Once each steps described in sections 5.3.2–5.3.6 achieved, the whole infrastructure described in section 5.3.1 creating the service emulated by the honeypot is now functional. Ensuring this is quite easy since each of the Elastic deployment node is monitored by Beats. Therefore, from the menu *Stack Monitoring* under the *Management* section of the Kibana tool, one can see an overview of the created Elastic cluster as well as the heath of each of its node. Figure 5.3.4 shows the expected Kibana view for the Elastic stack honeypot created in this chapter.

In order also to check if the deployed infrastructure works as excepted, one can observe if the logs coming from the Log Generator are indeed shipped to Elasticsearch by Logstash and are viewable in Kibana. To do this, first one needs to ensure that indexes are indeed created following the format introduced in line 27 of Code 5.8. This is verifiable from Kibana's *Stack Management* menu, using *Index Management* under the `Data` section. Figure 5.3.5 shows a list of indexes following the naming convention quoted above generated inside the Elastic cluster during several days.

[7] For more information about Metricbeat's CLI, see this website.

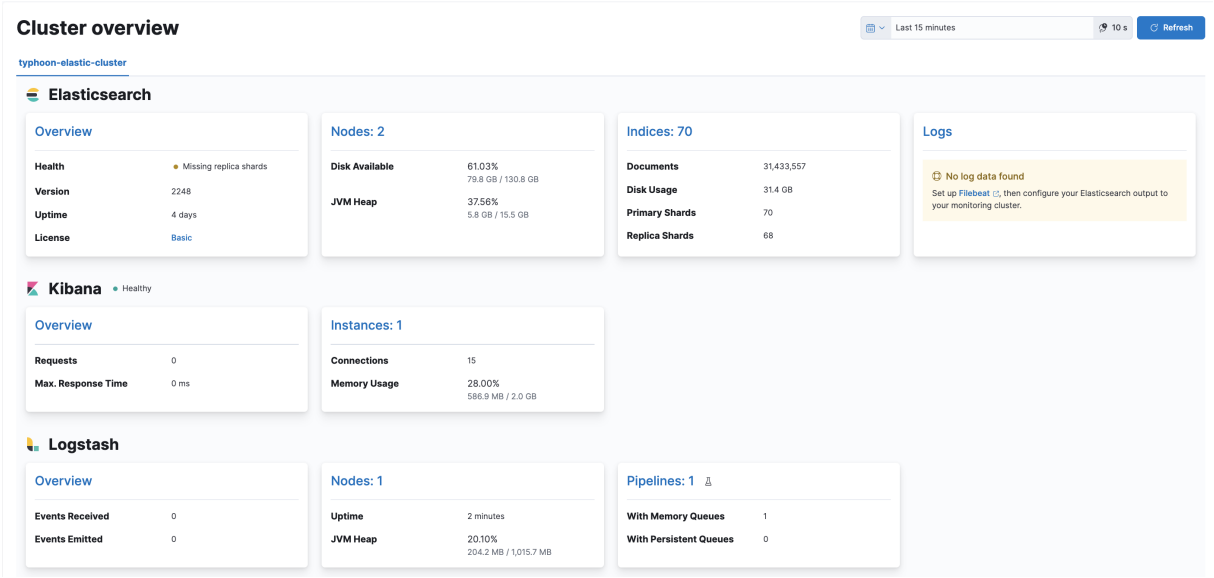


FIGURE 5.3.4

Overview from Kibana of the deployed ELK cluster named typhoon-elastic-cluster. This cluster is the exact same as the one described in 5.3.1, made of one Kibana instance as well as a Logstash one and two Elasticsearch nodes. All of these are monitored by Beats, enabling their surveillance, which allows this Kibana view.

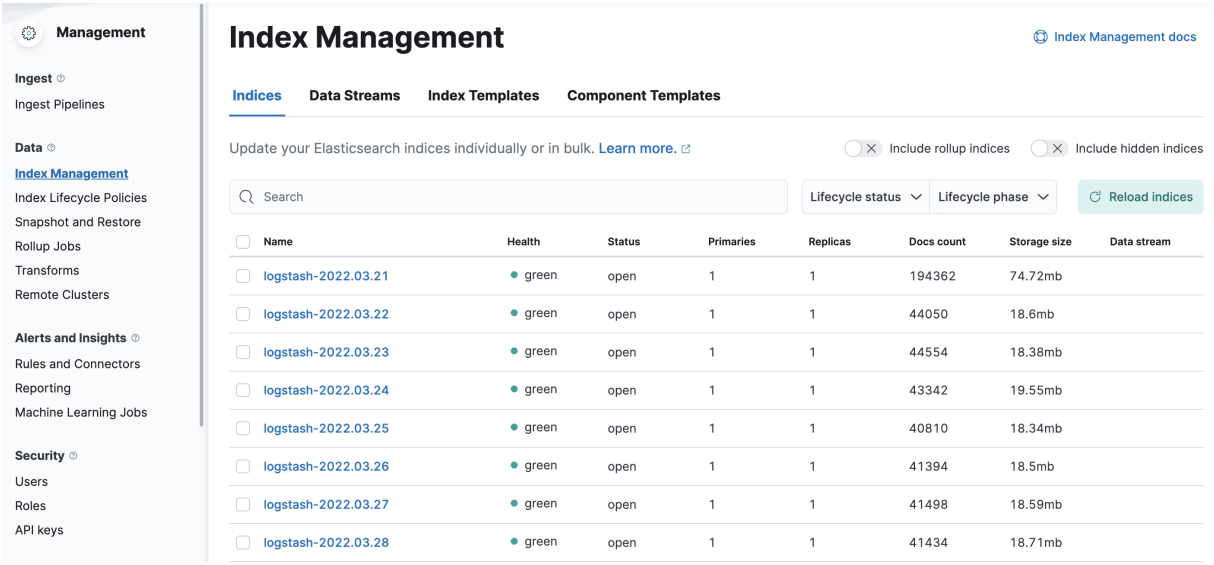


FIGURE 5.3.5

Overview from Kibana of the indexes containing logs generated by the Log Generator created inside Elasticsearch nodes of the deployed cluster.

Checking if the indexes are indeed created is already reassuring, however making sure that they indeed contain the logs created by Log Generator will complete the test of the deployed stack. To do this, it suffices to configure the *Log Stream* of Kibana's *Logs* menu in order to make it follows all the indexes created by the Logstash instance of the cluster. This is simply achieved by referencing `logstash*` as *Log indices* field from the setting of the *Log Stream* utility, which will load all of the log indexes in the cluster prefixed by `logstash`. Once this step achieved, the tab *Stream* under the *Logs* section from the *Observability* menu of Kibana should have a similar output than the one given in FIGURE 5.3.6. This confirms that the logs are indeed properly formatted, stored and viewable

inside the ELK stack deployment described in the sections supra.

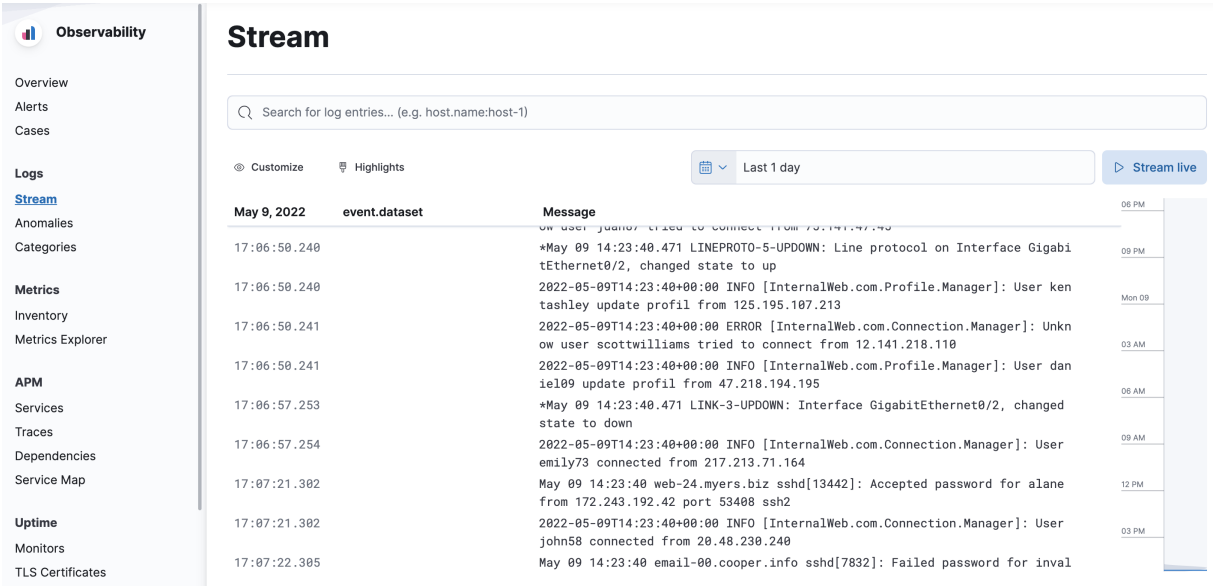


FIGURE 5.3.6
Overview from Kibana of the logs inside the indexes prefixed by `logstash`, created by the Log Generator application, detected and formatted by the Logstash node and stored inside the Elasticsearch nodes of the cluster.

After these few pseudo integration and system tests, one can be confident regarding the fact that the deployment of the ELK stack is indeed successful. Now that the mimicked services of the honeypot are deployed, its monitoring still needs to be configured in order to collect the data of interest as well as to avoid unpleasant surprises.

5.4 Monitoring the Honeypot

The monitoring strategy for the Elastic stack honeypot is very similar to the one described in section 4.4. The tools used for monitoring the honeypot behavior will remain the same, namely those of the SecureX suite. The following sections will describe how each step of the monitoring framework introduced in section 2.3 has been applied to the honeypot created in this chapter.

5.4.1 Comparative Basis Setup

As for the RDP honeypot created in chapter 4, the ELK stack honeypot has been deployed in the virtualized environment exposed to the public internet kindly made available by the University of Liège for this project (***cf.*** FIGURE A.2.1). For the same motivations than the ones given in section 4.4.1, each initial state of the four machines making the Elastic stack honeypot has been saved using the snapshot functionality of the hypervisor managing them.

Furthermore, as said supra, SecureX remains the tool used to monitor the honeypot, and therefore Secure Endpoint, Orbital and Threat Response will be also installed on each of the four devices. As explained in section 4.4.2, Orbital implements the Forensics Snapshots queries, allowing to collect a lot of informations summarizing a host state. Since once again this functionality will be used in the monitoring workflows for the Elastic stack

(***cf.*** FIGURE 5.4.1), one Orbital forensic snapshot by device of the honeypot is added to the comparative basis.

5.4.2 Monitoring Tools Deployment

The tools used to monitor the ELK stack honeypot are almost the same as the ones used for the RDP honeypot presented in chapter 4. Secure Endpoint connector as well as Orbital agents have been installed in each of the devices represented in FIGURE 5.3.1, being the tools at the heart of stack monitoring workflows. The installation steps for these utilities are the exact same as the ones given in 4.4.2, except that this time the connectors are the ones for Rocky Linux OS instead of Windows ones of course.

For Unix based systems, it is very simple to execute commands remotely on a device by using SecureX workflows and this even if it is part of the public Internet. Indeed, it can simply be achieved using the SSH protocol which has no real restriction regarding public or private deployment of a device, unlike his counterpart Remote Powershell on Windows devices. To collect more information about the system state than the ones that an Orbital query would be able to offer or the ones collected by Secure Endpoint, two CLI softwares for Linux based systems have been used, namely `top` and `iftop`.[90, 91] As it will be explained in section 5.4.3, the monitoring strategy presented here highly relies on monitoring network connections to the open port on the devices running the ELK stack honey deployment. Using the `top` utility allows to collect information about the device state for particular process (***e.g.*** CPU and memory usage for one of the Elasticsearch application just after a contact) while the second one, `iftop` is used to collect the network information of an host (***e.g.*** bandwidth taken by a connection to the host). The `top` utility is installed by default on Rocky Linux devices, however `iftop` should be installed, for instance with the default package manager of this OS.

5.4.3 Event Log

The global monitoring workflow used for monitoring the devices creating the Elastic stack honeypot is given in FIGURE 5.4.1. As the reader may notice, this workflow is very close to the one presented in section 4.4.3 for the monitoring of the RDP honeypot.

This workflow presented in FIGURE 5.4.1 will monitor processes using an open network port on a device. Therefore only the machines from the deployed infrastructure having an exposed port to the network should be targeted by this monitoring process. As it can be seen from sections 5.3.2-5.3.6, only machines of the deployment running the Kibana instance and the Elasticsearch ones are exposed to the network. This is achieved on the one hand by using their configuration and on the other hand by allowing the traffic reaching them to pass through the device firewall. Therefore, the fourth device of the honey deployment, the one running the Logstash node, will only be monitored by Secure Endpoint and not by any SecureX workflow. This is actually sufficient since the interest of the honeypot is to study potential threats targeting the honey Elastic deployment. However, these threats will most probably use network as an entry point to target the nodes of the honeypot. Since the firewall of the device running Logstash blocks every connection initialization attempt on any port of this machine, no workflow needs to be implemented to monitor them. Moreover, Secure Endpoint will still ensure that no malicious execution has taken place within this device.

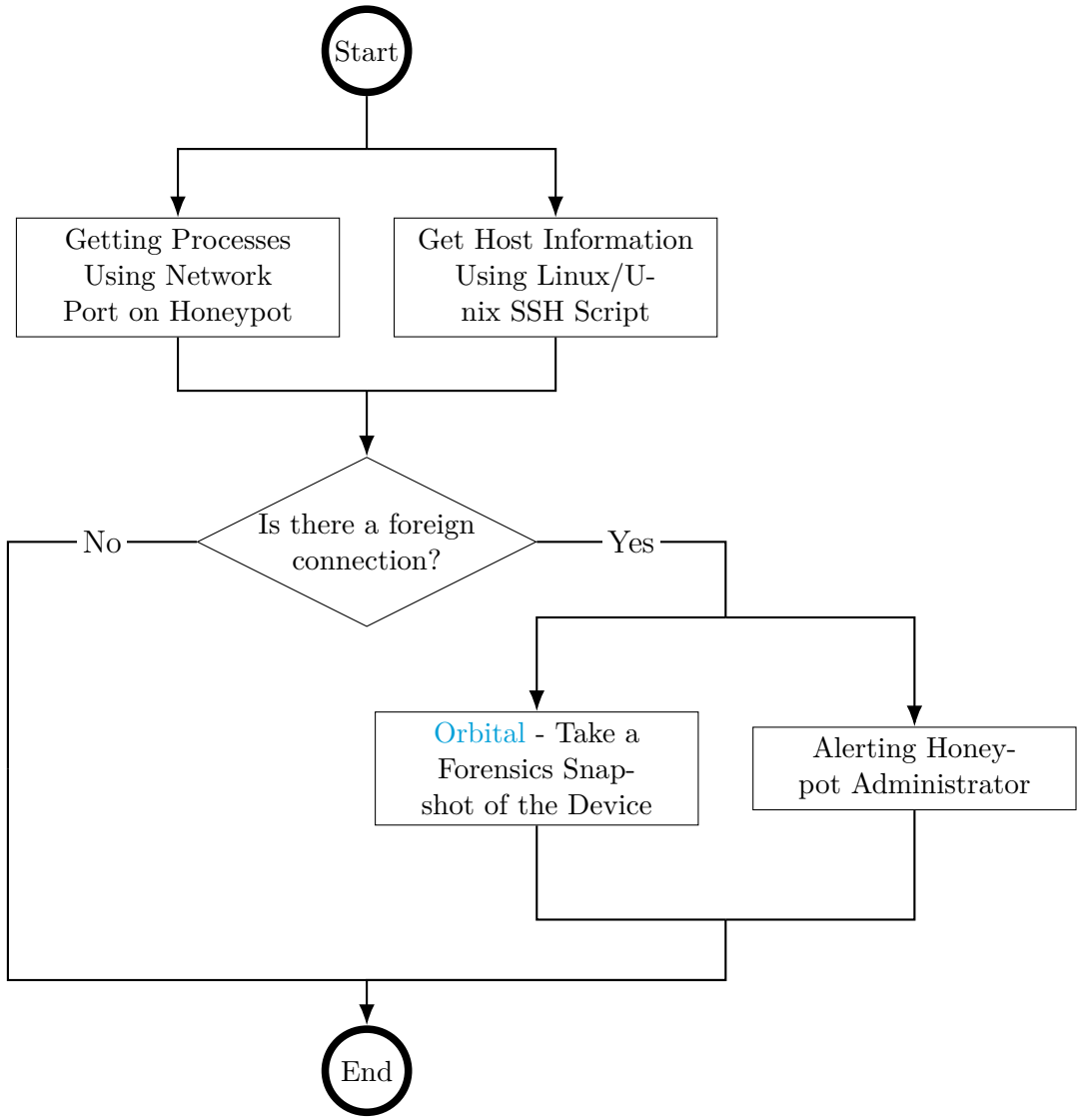


FIGURE 5.4.1

Global monitoring workflow for the created Elastic stack honeypot. This workflow is similar for each of the devices in the honey deployment having a network port open. The Getting Processes Using Network Port on Honeypot and Alerting Honeypot Administrator are sub-workflows similar to the ones depicted respectively in FIGURE 4.4.3 and FIGURE 4.4.4. In blue are highlighted the atomic operations calling one of the SecureX tool.

Even if the workflows are similar to monitor foreign connections established with Kibana and Elasticsearch services, one difference lies in the Orbital query made at its very beginning used to collect the connections for the process of interest. The workflow performing this query is still the same as the one given in FIGURE 4.4.3, however the queries themselves will be different depending on the services. Indeed, while Kibana will use HTTPS default network port 443 to handle connections, the Elasticsearch nodes will use ports 9200 and 9300. Therefore, two different Orbital queries will be used to detect processes using the open network port for both the services. The one relative to the Kibana connections is given in CODE 5.15 while the one for the Elasticsearch nodes is given in CODE 5.16. It can be emphasize here that the devices running the Kibana instance is monitor by its own workflow while the ones supervising the two other devices running the Elasticsearch nodes has been regrouped, but still follow the same steps.

As already mentioned in chapter 4, the SecureX workflows presented in this thesis are triggered based on a time event. For the Elastic honeypot presented in this chapter, each workflow has been triggered every three minutes, once again identified as the time for which an instantaneous connection remains in average in Orbital's `process_open_sockets` table.

```
SELECT pos.local_address, pos.remote_address, pos.local_port, pos.
    remote_port, COUNT(*) number_of_connexions,
CASE pos.family
     WHEN 1 THEN 'IPv4'
     ELSE 'IPv6'
END AS ip_protocol
FROM process_open_sockets pos LEFT JOIN processes p ON p.pid=pos.pid
WHERE pos.local_port=443 AND pos.remote_address NOT IN ("", "0.0.0.0", "
    127.0.0.1", "::", "::1","0")
GROUP BY pos.remote_address;
```

CODE 5.15

Orbital query getting the list of the connections using the HTTPS standard network port 443. The informations retrieved the local and remote IP address of the hosts involved in the connection as well as the port they use, the number of connections opened in parallel and the internet protocol version used. Of course this excepts connections coming from the host itself.

```
SELECT pos.local_address, pos.remote_address, pos.local_port, pos.
    remote_port, COUNT(*) number_of_connexions
FROM process_open_sockets pos LEFT JOIN processes p ON p.pid=pos.pid
WHERE (pos.local_port=9300 OR pos.local_port=9200) AND pos.
    remote_address NOT IN ("", "0.0.0.0", "127.0.0.1", "::", "::1","0", "
    ::ffff:a.a.a.a", "::ffff:b.b.b.b", "::ffff:c.c.c.c", "::ffff:d.d.d.d"
    )
GROUP BY pos.remote_address;
```

CODE 5.16

Orbital query getting the list of the connections using the Elasticsearch standard network ports 9200/9300. The informations retrieved are the local and remote IP address of the hosts involved in the connection as well as the port they use, the number of connections opened in parallel and the internet protocol version used. Of course this except connections coming from the host itself as well as the other services from the deployment connecting legitimately to the Elasticsearch nodes. The IP addresses of the nodes of the honeypot have been anonymized in this query.

One new action has been used to monitor the device on the Elastic stack honeypot compared to the workflow in FIGURE 4.4.2, it is defined as Get Host Information Using Linux/Unix SSH Script in FIGURE 5.4.1. This action allows to run a bash script remotly on a device and get back its output. It has been used to run the script given in CODE 5.17, getting first general host ressource consumption statistics as well as the one for a particular process and network information. Notice that in this piece of code, `elastic` is used to get the process ID of the service to monitor using `pgrep`, therefore this line allows to monitor an Elasticsearch service. However, simply changing this by `kibana` allows to monitor the Kibana service on a device, which has been used in this application monitoring workflow.

```
top -bn1 -p $(pgrep -d',' -f elastic)
iftop -tnPN -s 2
```

CODE 5.17

Bash script getting host and network state information using the `top` and `iftop` utilities. The first command get host health statistics as well as CPU/memory usage for the Elasticsearch process. The `pgrep -d',' -f elastic` command inside the first line can be replaced by the `pgrep -d',' -f kibana` one in order to get the exact same information but for a Kibana process running on the host. The second line probe the network for two second in order to get general statistics on the connections coming through the device using `iftop`.

As for the honeypot presented in chapter 4, if a foreign connection is detected on the host (***i.e.*** if the Orbital query made is not empty), then the workflow will take a forensics snapshot of the device and in parallel send an alert using Webex to the honeypot administrator, still using the workflow depicted in FIGURE 4.4.4. If no connection is detected, then the workflow simply ends.

By using three variations of the workflow depicted in FIGURE 5.4.1 for the three devices exposed to the network in the deployment, the connections initiated with the Elastic stack honeypot can therefore be easily and efficiently monitored. All of the four devices will be additionally monitored by Secure Endpoint, which ensures even more visibility.

5.4.4 Critical Event Alerting

The same rules as the ones applied in section 4.4.4 are also valid for this deployment. As explained in section 2.3.4, every event or change detected in a honeypot should be treated as critical. In this monitoring strategy as for the previously presented one, each connection to the Elastic stack deployment triggers an alert directly transferred to the honeypot administrator using Webex. However, these alerts will be a bit more complete than the ones of the section 2.3.4, since they will also contain the output of the script depicted in CODE 5.17, giving more information about the device state during a triggered event. A typical alert sent using the monitoring workflow presented in this section is given in FIGURE 5.4.2.

5.5 Elastic Stack Honeypot Classification

To begin the classification, it seems obvious that once again the deployed honeypot is made of several honey services, mimicking the behavior of an ELK stack deployment. Honeytokens are also deployed at several level of the infrastructure and will be reviewed in section 5.5.3. However, the field of allowed action using the deployed machines is limited and does not offer the same functionalities as a whole OS, therefore no honey systems has been deployed here.

5.5.1 Purpose

As already mentioned in section 5.4.1, the Elastic stack honeypot has been placed in the public Internet area provided by the University of Liège (***cf.*** FIGURE A.2.1). Regarding the placement of the devices composing the deployed infrastructure and comparing it to the FIGURE 1.2.1, the honeypot can be classified as research honeypot. Furthermore, since the goal of this device is to better understand the actions of the black hat community

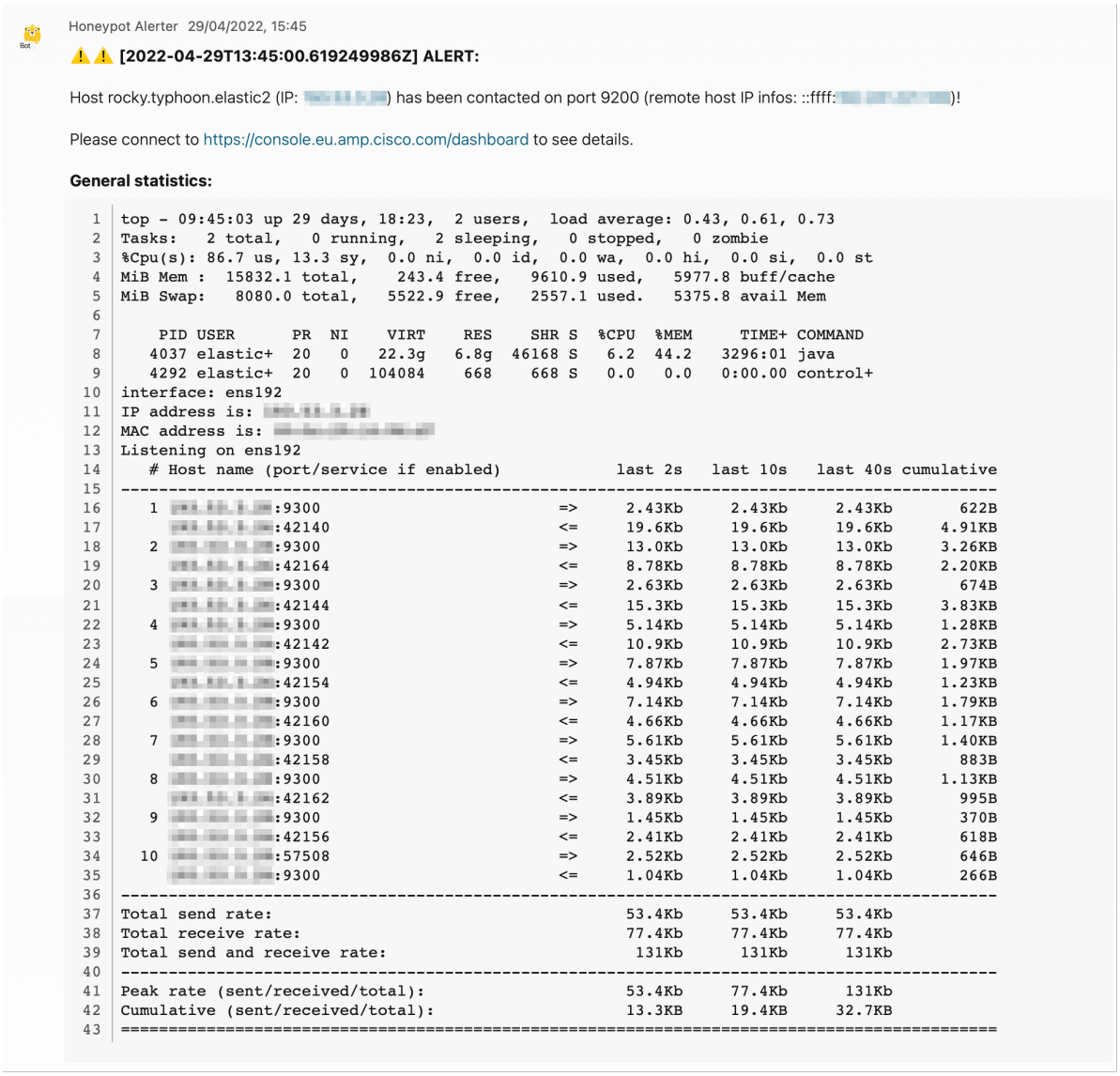


FIGURE 5.4.2
Alert sent using the ELK monitoring workflow to the honeypot administrator.

regarding the services it mimics, classifying the honeypot as part of the research family seems to fit with its purpose.

5.5.2 Interactivity

Certainly, the deployed honeypot emulates several services while still not offering full OSes functionalities.[8] As the reader may have noticed, the section 5.3 contains much more considerations than the ones made for the proof of concept RDP honeypot in section 4.3. If one refers to the graph given in FIGURE 1.3.1, since the intelligence level provided for this deployment is relatively consequent while not being extremely complex, it seems

[8]The scientific literature often adopts another terminology while describing the honeypots similar to the one presented in this chapter. These are defined as *honeynets*, being networks of multiple honeypots.[3, 13, 32, 33, 68, 92] In this thesis, it has been decide not to differentiate between honeypots and honeynets in the adopted terminology. Indeed, the definition introduced in section 1.1 remains valid for honeypots made of several devices, therefore introducing a new theoretical concept did not seem interesting.

correct to affirm that the interactivity that should result from the deployment should also be medium. In this writing, this honeypot has therefore been classified as a medium level interaction one.

5.5.3 Honeytokens

For this deployment, the reader may have noticed that several honeytokens have been defined towards the configuration given in section 5.3. This section will highlight various considerations made while generating them.

5.5.3.1 Fake University Spin-off Data

Inside each of the configuration files presented in section 5.3, the word Typhoon appears often to define service names. As already briefly mentioned in section 5.3, to increase the credibility of the created honeypot, Typhoon is a 100% fake university spin-off using the deployed Elastic stack as infrastructure monitoring tools. Indeed, deploying an ELK stack in the public Internet is not unreasonable as stated in section 5.3.3.2, however this infrastructure should appear as serving to someone to appear as legitimate. This is namely why Typhoon has been created, in order to increase the credibility of the deployed honeypot.

Typhoon is of course a honeytoken and to be even more precise, a data level honeytoken. This honeytoken is actually close to the example presented for the property 3.4 from section 3.2. Even is this fake data respects each of the properties 3.1-3.7, the one guarantees the most is for sure detectability (***i.e.*** property 3.4). Indeed, inside the real infrastructure in which the honeytoken is deployed, every hosts, services or data flagged with this name can be easily detected as being part of the Elastic stack honeypot.

5.5.3.2 Kibana DNS Name

As already discussed in section 5.3.5, a DNS entry has been created for the Kibana server. Once again, this data is actually a network level honeytoken which fits completely with the example given in their definition from section 3.3.1.

Behind this entry creation lies several motivations. First, one can see that the URL `https://kibana.typhoon.uliege.be/` is part of the university domain (since it is suffixed by `uliege.be`) and that the Typhoon word appears in it (which ensure properties 3.4 and 3.6) as well as the Kibana one. With that, the honeytoken ensures property 3.1 since this URL could definitely be found in production, resuming the service as well as the company it serves. Furthermore, indexing the created Kibana service inside a DNS infrastructure should increase its conspicuousness (***cf.*** property 3.3). By using a detectable honeytoken, the interactivity level on the device is therefore expected to increase too. This assumption will be discussed in 7.3 of this writing. Regarding the attractiveness level (***i.e.*** property 3.2) for a Kibana service, this is actually tested by the honey deployment and will be discussed also in part III of this writing.

5.5.3.3 Fake Infrastructure Logs

Each logs created by the Log Generator presented in section 5.3.2 can be also considered as a data honeytoken. These logs are generated to increase the credibility of the honeypot. Indeed, an ELK stack monitoring nothing in a production environment would be much less attractive that one of its counterpart containing sensitive information on events triggered in an infrastructure. These honeytokens are therefore designed to be

attractive for the black hat community while also appear as legitimate (***i.e.*** ensure properties 3.1 and 3.2). However, each log follows the standard form of well-known services and highly relies on random generator to create the variables that may appear in their structure. For instance, CODE 5.18 gives a sample of an output that can be generated by the Log Generator application. In this sample, typical SSH authentication, application and Cisco IOS logs are generated following their standard format. However, by finding these in a real infrastructure, it will be difficult to define them as not legitimate which does not really respect honeytokens properties 3.4 and 3.6. This might be seen as a limitation of the deployment as it will be discussed in chapter 8. Nevertheless, these logs should normally transit inside the controlled honeypot environment depicted in FIGURE A.2.1, therefore the risk of interference should remain low.

```
2022-05-09T14:15:42+00:00 INFO [InternalWeb.com.Connection.Manager]:
    User gonzalezbrett connected from 215.238.107.58
*May 09 14:15:42.401 SYS-5-CONFIG_I: Configured from console by console
2022-05-09T14:15:42+00:00 DEBUG [InternalWeb.com.Transaction.Manager]:
    Starting transaction for session 7
    f8414858bb9303e953e19c7cf047fc521070300
2022-05-09T14:15:42+00:00 DEBUG [InternalWeb.com.Transaction.Manager]:
    Starting transaction for session 125935
    adc8821f6f4461f29e1f61c862302fbbf8
2022-05-09T14:15:42+00:00 INFO [InternalWeb.com.Connection.Manager]:
    User daniellejimenez connected from 30.137.52.186
2022-05-09T14:15:42+00:00 ERROR [InternalWeb.com.Connection.Manager]:
    Unknow user marydaniels tried to connect from 35.72.162.133
May 09 14:15:42 db-53.nolan-henry.com sshd[3432]: Accepted password for
    joshua44 from 110.54.89.58 port 19438 ssh2
*May 09 14:15:42.400 SYS-5-CONFIG_I: Configured from console by console
2022-05-09T14:15:42+00:00 ERROR [InternalWeb.com.Connection.Manager]:
    Unknow user hollandjamie tried to connect from 104.59.202.150
2022-05-09T14:15:42+00:00 ERROR [InternalWeb.com.Connection.Manager]:
    Unknow user angelastevens tried to connect from 213.116.225.80
2022-05-09T14:15:42+00:00 DEBUG [InternalWeb.com.Transaction.Manager]:
    Starting transaction for session 2380
    e3c830dc06c824c68a62f8a169b16c008137
*May 09 14:15:42.400 SYS-5-CONFIG_I: Configured from console by console
```

CODE 5.18

Example sample of logs generated by the Log Generator application presented in section 5.3.2. Inside this sample can be found typical SSH authentication, application and Cisco IOS logs following their standard structures. Variable data inside these have been generated using the Faker[86] Python library, generating fake data of any kind.

5.6 Conclusion

In this section has been presented a honeypot simulating an Elastic stack deployment. Each steps of the honeypot creation as well as its monitoring strategy has been detailed. The output of these is a medium level interaction research honeypot whose purpose is to help a company having a better visibility on one of their daily used application.

Part III

Results and Discussion

Remote Desktop Protocol Data Analysis

This chapter presents the data collected by the remote desktop protocol honeypot created in chapter 4. The data set structure will be presented first, then each of its variables will be inspected in more detail and finally a small discussion about the observations made will be proposed.

6.1 Data Set Review

Before starting to enumerate the data set's records structure, it is important to underline that those have been generated by only one of the honeypot created chapter 4. Indeed, recall that in this part of this thesis, two devices have been described, the first one placed in Cisco's lab (***cf.*** FIGURE A.1.1) and the second one deployed inside the public Internet area (***cf.*** FIGURE A.2.1). However, during its entire deployment period, the RDP honeypot placed inside the laboratory of Cisco didn't produce any data. As already mentioned in section 4.5.1, this location is supposed to be highly aseptic and thus, not detecting any illegitimate traffic to a device placed in this space is reassuring and was expected. Therefore, every data presented here comes only from alerts triggered by the monitoring framework presented in section 4.4, for the RDP honeypot placed inside the public Internet area made available for this thesis (***cf.*** FIGURE A.2.1).

Now that the source of the analyzed data set has been clarified, its structure can be presented. Each record from this data set is made of five fields listed below.

- **Alert Timestamp:** the timestamp of the alert in RFC 3339[93] format `YYYY-MM-DDTHH:mm:ss.SSSSSSSSSZ` (***e.g.*** `2022-04-26T08:45:01.047004316Z`);
- **IP Address:** the IP address of the source initiating the RDP connection with the honeypot (***e.g.*** `192.241.221.151`);
- **AbuseIPDB Link:** the URL of the research into AbuseIPDB database for the IP address initiating the connection with the honeypot (***e.g.*** `https://www.abuseipdb.com/check/192.241.221.151`);
- **Region:** the region where the IP address is located according to AbuseIPDB and MaxMind records (***e.g.*** San Francisco, California);
- **Country:** the country where the IP address is located according to AbuseIPDB and MaxMind records (***e.g.*** United States of America);

From these five fields, three have been defined based on another. Indeed, AbuseIPDB[1] has

[1] See this website.

been used to track the IP address of the RDP connections initiators with the honeypot. This website is useful to check if an IP address has been defined as malicious by the community or even reporting some of them. Using this tool, the link of the query used in AbuseIPDB for checking a particular IP address as well as both the region and the country of the location of this address have been derived from the IP address field. The geolocation of IP addresses has also been verified using MaxMind[2], more commonly used among the scientific community.

Now that the structure of the collected data has been presented, the following sections present the analysis of each field detailed supra. Only the two fields that are not redundant (***i.e.*** the timestamp and the IP address ones) will have a dedicated analysis part in section 6.2, since the other three are derived from those.

6.2 Analysis of the Different Data

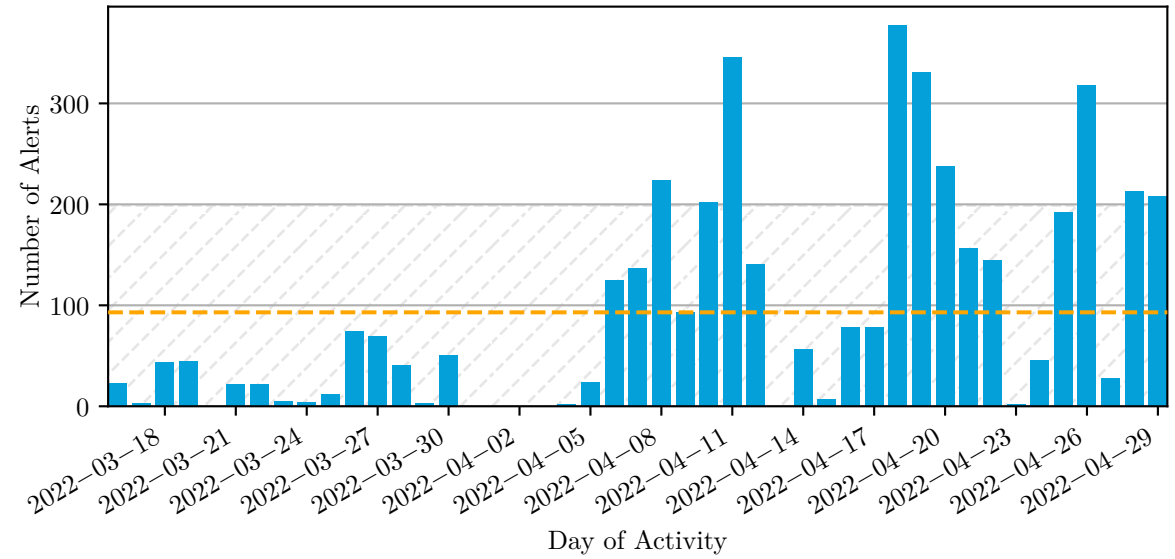
6.2.1 Timestamps

The timestamps of the records collected by the RDP honeypot in the public Internet are the ones of the beginning of the monitoring workflow for the device (***cf.*** Figure 4.4.2). Indeed, in SecureX, each workflow starting time is recorded inside the workflow output variable `Start time` of type `DATE`. This variable has been used as alert timestamp for any connection attempt with the monitored honeypot. It is worth mentioning that these metrics are given for the Zulu time zone (equivalent to the Universal Time Coordinated, ***a.k.a.*** UTC) as indicated by the `Z` in the data. All honeypots placed either in Cisco's lab or in the dedicated public Internet space (***cf.*** appendix A) are however deployed in a Central European Summer Time (CEST) location for the whole deployment duration.[3] For information completeness, one can note that this timezone is 2 hours ahead of the UTC one. Nevertheless, the data analysis should of course not be affected by this jet lag, since it simply represent a translation in the analyzed data. Figure 6.2.1 graphically presents the time metrics collected by the RDP honeypot placed in the public Internet area. Inside this figure, several histograms can be found, presenting the key metrics contained inside the timestamp data. Each of them is reviewed in the following paragraph.

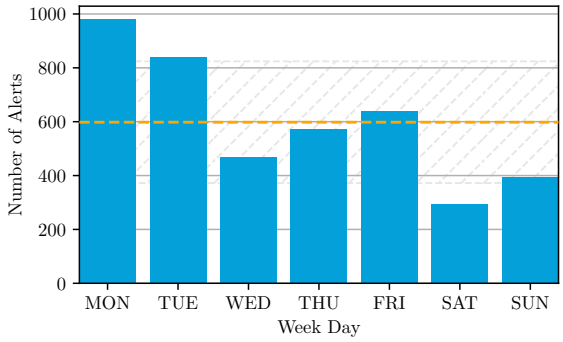
Figure 6.2.1a presents the number of RDP connections initiated with the honeypot by its day of activity. From this figure, one can see the distribution of the 4187 observed connections over a period of 44 days. Either from the graph or from the value in Table 6.2.1, it can be observed that on average, the honeypot has been contacted 93 times per day. However, by inspecting Figure 6.2.1a, it seems obvious that the number of connections varies greatly from this mean value. This is confirmed by the high value of the standard deviation of these metrics compared to the mean one (***cf.*** Table 6.2.1).

Focusing on weekly metrics and not the ones for the whole period of data collection anymore, one can see the number of connections by week day for the whole honeypot activity period in Figure 6.2.1b. From this graph can be noticed that the weekend days are the ones for which the honeypot activity is the lowest while it has the highest activity on Monday (around 1.6 times the mean activity value).

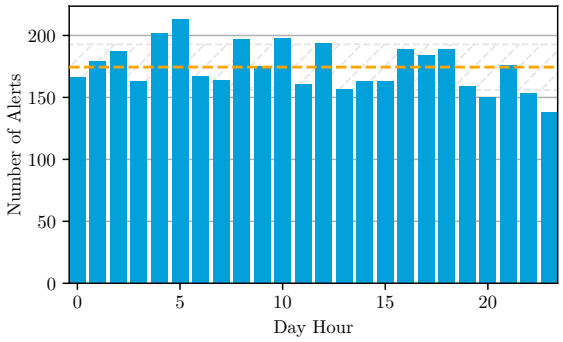
[2] See this website.
[3] Belgium may also be in the CET zone, ***i.e.*** CEST-1.



(a) Number of connections initiated with the honeypot per its day of activity.



(b) Number of connections initiated with the honeypot per week day.



(c) Number of connections initiated with the honeypot per hour of the day.

FIGURE 6.2.1

Connections initiated with the RDP honeypot placed in the public Internet area presented in chapter 4. These graphs present the global timestamp metrics of the observed alerts. In each histograms, the mean value of the number of connections by the respective metrics is represented by a dashed orange line. The gray dashed hashed area gives the standard deviation area around the mean of the data. These constant values are also given numerically in TABLE 6.2.1.

	By Day of Activity	By Hour	By Week Day
Mean number of connections	93	174.458	598.14
Standard deviation of the number of connections	105.809	18.401	226.062

TABLE 6.2.1

Mean and standard deviation of the number of connections for day, hours or week day activity of the RDP honeypot placed in the public Internet (as represented in FIGURE 6.2.1).

Again readjusting the center of attention from the timestamp analysis to focus this time on the daily distribution of the number of connections, the number of alert by the hour in the day can be studied. FIGURE 6.2.1c therefore gives the count of RDP connec-

tions initiated with the honeypot depending on the timestamp hour. This distribution has the lowest standard deviation (***cf.*** Table 6.2.1) of the time data analyzed in this section, which translates to the fact that the number of connections by day hour are close to their mean value, as it clearly appears in Figure 6.2.1c. Since the data by hour are close to each other, no daily pattern can therefore be identified from this data set.

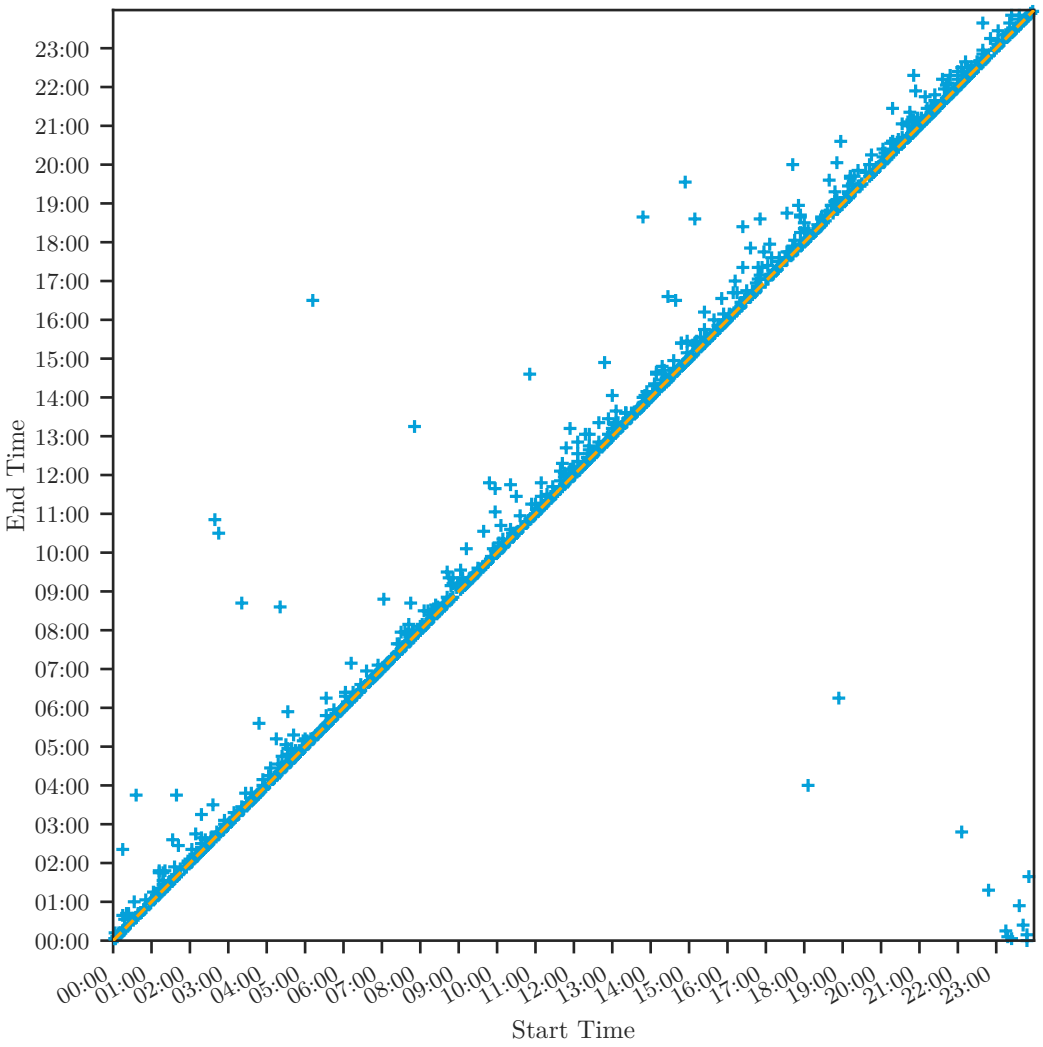


FIGURE 6.2.2

Connections initiated with the RDP honeypot placed on the public Internet represented as pairs of timestamps for a 24 hours × 24 hours space. The first element of the pair, represented in the x-axis is the start time of the connection while the second timestamp, given by the y-axis, is its end time. The orange dashed line gives the identity of the graph, representing instantaneous connections, ***i.e.*** connections for which the start and end timestamps are the same.

In this case study, raw data can be extracted from timestamps (***e.g.*** day, hour, minute, ...) in order to find interesting patterns as done in the paragraphs above. However, another important metric that can be derived from this data type in order to study general behaviors regarding the honeypot is the connection time. As already explained in section 5.4.3, the SecureX workflow used to monitor the RDP honeypot is triggered using a time based event. This event launches the workflow every three minutes in this use case. Nevertheless, from experience, it has been noticed that some failures can happen between workflows, coming from either the monitored device or from the SecureX platform and

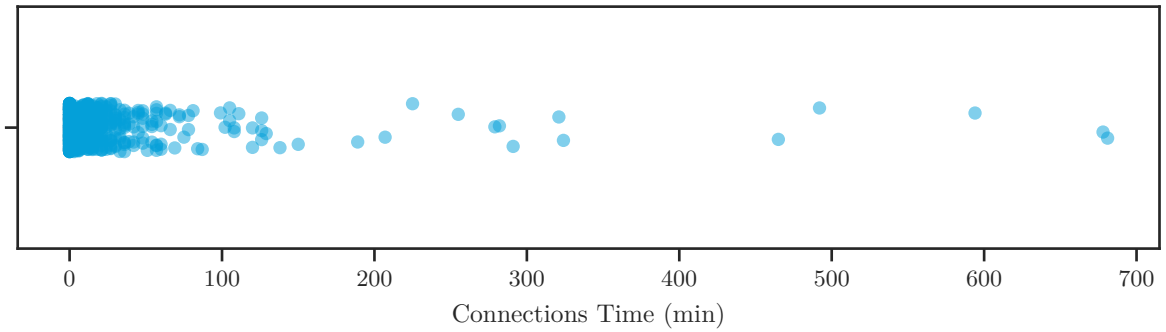
therefore preventing the collection of precise data. To overcome these small inconsistencies that may appear in the data set, a connection between the RDP honeypot and a host identified by its IP address has been defined as the longest difference between two alert timestamps for a same IP address for a serie of them uninterrupted by more than 15 minutes. Based on this assumption, the connections between the honeypot and foreign hosts can therefore be defined by a pair of timestamps, the first one giving the initial time for which an alert arose for a particular IP address and the second one providing the last alert triggered by this same IP. In between, any number of alerts for this IP address can exist, provided that each of them is not separated from the previous one by more than 15 minutes. FIGURE 6.2.2 shows the relationship between the starting times of the RDP connections experienced by the honeypot and their respective end times. The reader may have noticed that this graph covers a 24 hours $\times$ 24 hours space, meaning that no connection lasted more than one day. In this figure, the graph identity has also been added, representing the connections not having lasted more than a single timestamp and therefore being considered as instantaneous. One can see that the distribution is actually close to this identity and therefore that most of the measured connections have not been very long considering a scale of one day.

Another way to visualize the distribution of the connections between the honeypot and a host duration is by using the well known strip and violin plots. The ones relative to the connection time defined above are respectively depicted in FIGURE 6.2.3a and FIGURE 6.2.3b. By analyzing these graph, one can see that the connection duration varies from 0 to 681.005 minutes (***i.e.*** around eleven hours and twenty-one minutes). However, most of the connections are located around the same area of the graph, which has been highlighted in FIGURE 6.2.3c. The box plot of the distribution is also represented in the density curve depicted in this figure, with the rectangle showing the ends of the first and third quartiles (respectively noted $\mathcal{Q}_1$ and $\mathcal{Q}_3$) and the central white dot giving the median (also called second quartile, ***a.k.a.*** $\mathcal{Q}_2$). From this, one can see that the minimum, $\mathcal{Q}_1$, and the median are all equal to 0. This translates to the fact that at least 50% of the connections did not last more than one alert. From the location of $\mathcal{Q}_3$, one can deduce that 75% of the connections have a duration between 0 and around 10 minutes (9.0006 minutes precisely). For information, the high outliers from this distribution raise the mean value of the connection duration above the one of $\mathcal{Q}_3$, this one being of 12.476 minutes.

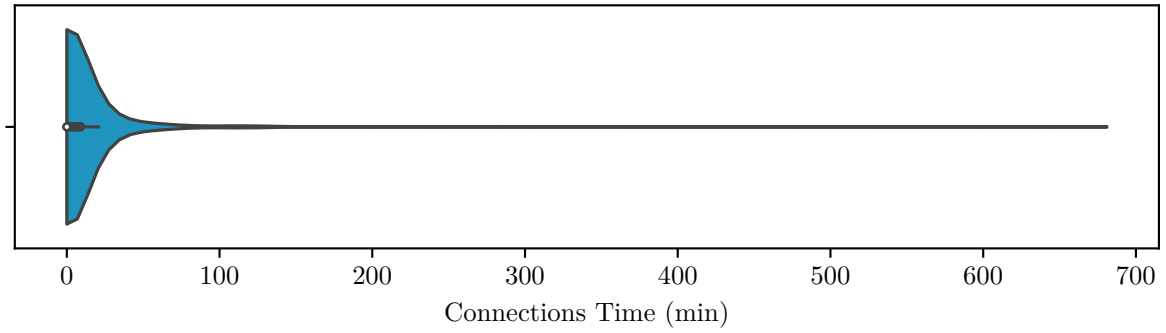
6.2.2 IP Addresses

The second analyzable metric in the data set presented in this chapter is the IP addresses of the host having initiated the connection with the honeypot. As already mentioned, the AbuseIPDB website has been used in order to briefly analyze these IP addresses, namely by getting their world location[4] but also to check what the website calls the *Confidence of Abuse* percentage. This percentage is a score based on user reports and their age regarding particular IP addresses. It defines the level of confidence with the one an IP address can be defined as entirely malicious. For the record, each IP address in the data set studied here was found in AbuseIPDB's database around the time of its contact with the honeypot and therefore has been reported as malicious by the community using this platform.

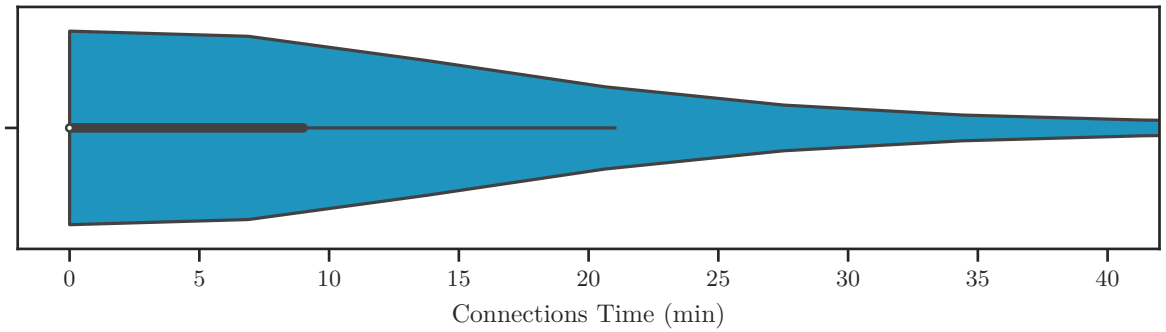
[4]From AbuseIPDB website, it seems that IP addresses ISP, usage type, and location are defined through a partner, IP2Location. These locations has also be double checked using MaxMind, as already explained.



(a) Strip plot of the univariate connection time data represented in minutes.



(b) Violin plot of the univariate connection time variable represented in minutes for the total range of data (***i.e.*** for the space $[0, 700]$).



(c) Focus on the main data distribution part of the violin plot given in FIGURE 6.2.3b. This plot still represents the distribution of the univariate connection time variable in minutes, this time for the limited range of data $[0, 40]$.

FIGURE 6.2.3

Strip and violin plots representing the distributions of the univariate duration of connection variable for different domains. The corresponding box plot of this distribution is also given in FIGURE 6.2.3b and FIGURE 6.2.3c, showing the values $\mathcal{Q}_1 = \mathcal{Q}_2 = 0.0$ and $\mathcal{Q}_3 = 9.0$.

FIGURE 6.2.4 illustrates the geographical location of the IP addresses as defined by the websites presented above and TABLE 6.2.2 shows the numerical values depicted in the figure. From these, it can be seen that even if the connection initiators come from all around the world, a lot of them are coming from central Europe, from the Netherlands to be more precise. The United States follow with nevertheless much less connections initiated and then India comes in third position.

Another interesting metric that can be studied while analyzing IP addresses is the emerging of subnets among them. Indeed, it is well known that IP addresses belonging

FIGURE 6.2.4
World location of the IP addresses having contacted the RDP honeypot placed in the public Internet area.

to a same subnet are addressed with an identical most-significant bit-group in their IP addresses. Identifying subnets in the data set would therefore allows to identify groups of malicious devices and not only a single one. The identified sub-networks inside the data set are given (alongside with the group of IP addresses belonging to them) in TABLE B.1.1 from appendix B. From this analysis, it can be observed that actually, 86.131% of the data set's IP address are members of an identified sub-network of prefix length above 8 bits.[5] When considering only IP addresses being part of a sub-network with a prefix size above or equal to 24 bits, 56.628% of them are part of a subnet.

When analyzing IP addresses, it is also a good approach to think in terms of autonomous systems. An autonomous system is a network regrouping IP prefixes under the same administrative control and each AS is identifiable by its globally unique autonomous system number (ASN).[94] It is therefore possible to determine the AS responsible for each IP address in the data set studied in this chapter. To perform this mapping between the ASN of each autonomous system owning each IP addresses of the data set, the well known Team Cymru[95] service has been used in this thesis. This mapping between ASN and IP addresses has also been double checked using data sets coming from the Center for Applied Internet Data Analysis (CAIDA).[96][6] Once the ASN of the responsible AS for each IP address obtained, a small analysis of its relations with other ASes has been performed, once again using one of the CAIDA data sets[96], defining autonomous systems relationships. Regarding the data set used to define links between AS, two types of relations can be found being the well known costumer-provider and peer-to-peer ones. The ASes defined for the studied data set have been classified based on these relations into three different categories

[5]This prefix length has been chosen since 8 is the lowest prefix length appearing in the CAIDA data set regrouping the IP prefix owned by the Internet autonomous systems (ASs). This data set is available in the `data` folder on the GitHub of this thesis.

[6]These data sets are available on the GitHub repository of this thesis, in the `data` folder.

Country	Continent	Number of Connection	Proportion of Connections
Netherlands	Europe	2422	57.846%
United States of America	North America	479	11.44%
India	Asia	336	8.025%
Russia	Europe	282	6.735%
Luxembourg	Europe	141	3.368%
Bulgaria	Europe	104	2.484%
Italy	Europe	85	2.03%
China	Asia	67	1.6%
United Arab Emirates	Asia	66	1.576%
Myanmar	Asia	47	1.123%
Turkey	Asia	38	0.908%
Mexico	North America	27	0.645%
Germany	Europe	27	0.645%
Taiwan	Asia	23	0.549%
United Kingdom	Europe	17	0.549%
Ukraine	Europe	15	0.358%
Indonesia	Asia	5	0.119%
Singapore	Asia	5	0.119%
Egypt	Africa	1	0.024%

TABLE 6.2.2

Number of connections initiated with the RDP honeypot placed in the public Internet area and the proportion of the total number they represent by country, as depicted in FIGURE 6.2.4.

- **Stub:** an autonomous system having only customer relationships;
- **Tier-1:** an autonomous system having only peer of provider relationships;
- **Transit:** an autonomous system having any type of relationships.

FIGURE 6.2.5 gives the classification of the ASes studied in this section. One can see that most of the autonomous systems are transit ones, while the other two categories still represent non negligible part of the data. FIGURE 6.2.6 represents the number of connections initiated with the honeypot by hosts with IP addresses coming from AS of different types defined supra. One can see from this figure that even if the Tier-1 is the less represented type among the ASes found in the data set, IP addresses from this AS type are the one having initiated most of the connections with the RDP honeypot. Stub category, although representing almost 25% of the data set's ASes, is only responsible of 627 connections (14.975% of the total number of them for this honeypot).

TABLE 6.2.3 presents the top five of the AS owning the largest number of IP addresses in the data set. Still from this table, it can be observed that the transit AS 9829 owns the larger group of IP addresses having initiated a connection with the RDP honeypot, representing 35.036% of the total 137 ones.

Still regarding IP addresses, one last experiment has been conducted during the period of creation of the data set analyzed here. At the beginning of this period, 10 IP addresses among the most *aggressive* ones have been blocked using the RDP host firewall until April 4, 2022 at 15:26:02 CET. The goal of this experiment was to observe if IP addresses would have re-initiated a connection after being blacklisted. From the data set, two of these ten blacklisted IP addresses (***i.e.*** 20% of them) reached back the honeypot after being whitelisted again on the device. These two IP addresses took 7 days, 19 hours and 58 seconds and 7 days, 16 hours, 30 minutes and 58 seconds (***i.e.*** a mean time of 7 days, 17 hours, 45 minutes and 58 seconds) to initiate a new connection with the honeypot after

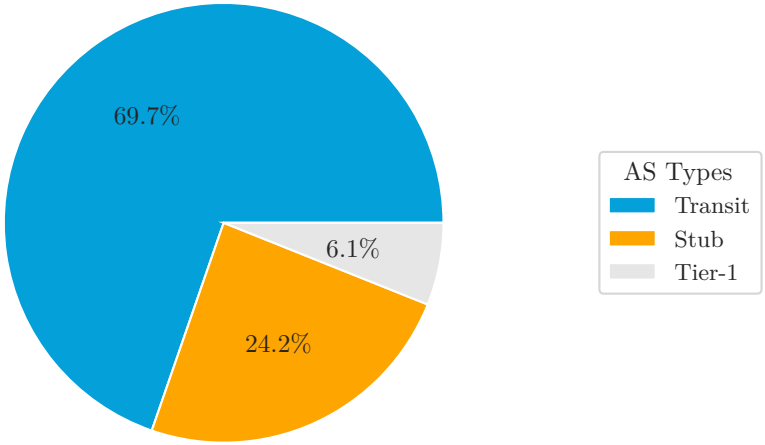


FIGURE 6.2.5

Proportions of autonomous system types (***i.e.*** stub, tier-1 or transit) present among the AS derived as owner of the IP addresses in the data set.

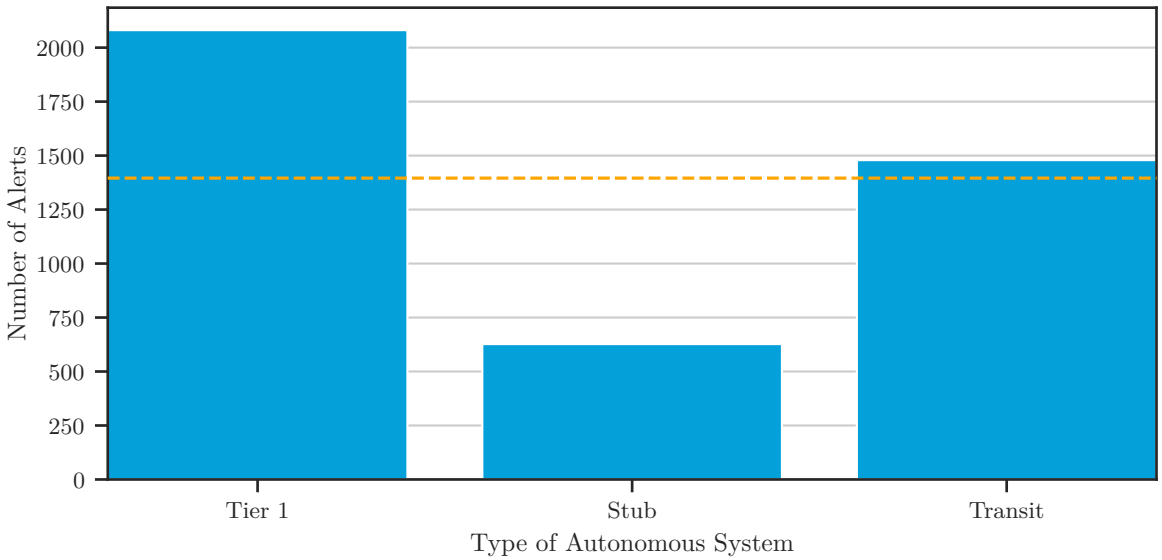


FIGURE 6.2.6

Number of alerts in the data set by autonomous system types (***i.e.*** stub, tier-1 or transit). The mean value of the number of connections by AS type is represented by a dashed orange line.

AS Number	AS Type	IP Addresses	Number of IP Addresses	Proportion of IP Addresses
9829	Transit	117.239.224.0/22, 117.254.112.0/20	48	35.036%
3257	Tier-1	181.214.206.0/24	17	12.409%
174	Tier-1	191.96.168.0/24, 191.96.160.0/19, 195.78.54.0/24	14	10.219%
14061	Transit	192.241.192.0/19	11	8.029%
212238	Stub	212.102.35.0/24, 212.102.32.0/19	10	7.299%

TABLE 6.2.3

Five first autonomous systems responsible of the higher numbers of IP addresses in the data set for the RDP honeypot placed in the public internet area. Only the subnet addresses owned by the respective AS in which the IP addresses in the data set can be found are given (***cf.*** TABLE B.2.1 for the exact list of address).

having been blacklisted.

6.3 Discussion

From the data analysis presented in section 6.2, several clues indicate that the RDP honeypot was targeted mostly by password guessers.[7] Indeed, the fact that first RDP is only an authentication mechanism to connect to a Windows device and that the contacts initiated with the honeypot lasted over time, thus were not instantaneous, indicates that several credential combinations was tried to gain access to the host. However, since the tried credentials was not stored (***e.g.*** using key logger), it is difficult to classify the type of password guessers that the honeypot faced during its deployment period. Enriching the data set with such data might be a good idea for future deployments, as it will be discussed in chapter 8.

Regarding the geolocation of the hosts having initiated a contact with the honeypot, it is important to walk on eggshells while using the analysis presented in section 6.2.2. In the era of Cloud service providers (CSPs), virtual private networks (VPN) services and anonymous proxy among others, finding the location of an IP addresses does not mean finding the real end host having initiated the connection anymore. Furthermore, other tools and methods have spread among the black hat community, allowing insiders to cover their tracks for while already, like IP spoofing or botnet usage. Moreover, IP address geolocation may appear less precise and remains a difficult field of research. However, locating the services quoted above can still be useful to detect certain trends regarding the method used to avoid detection (***e.g.*** use application in CSP instead of personal computers).

[7] In this thesis, a password guesser refers to a program conceived for trying to guess credentials used in an authentication mechanism. Among password guessing techniques can be found the well known brute-force attacks, dictionary attacks, credential stuffing attacks, birthday attacks, ...

The same comment applies for AS identification. Indeed, even if the mapping between an IP address of the initiator of a connection with the honeypot and the AS designated as its owner is normally indisputable and can be considered as a strong metric, their is no way to ensure that the connection request was actually initiated inside this AS. In fact, some autonomous systems, 19.2% of them according to the CAIDA Spoofer project[97], still allow IP spoofing. This means that, for any connection in the studied data set, the IP addresses of the hosts having contacted the honeypot could potentially impersonate an IP address owned by another autonomous system than the one identified as its owner, provided that the connection packet has been forged in an AS not blocking IP spoofing. Detecting IP spoofing based on the metrics present in the data set presented in section 6.1 is nonetheless not possible, which highlight a limit of the collected data. IP spoofing is basically used either to launch (D)DoS attacks, man-in-the-middle attacks, to penetrate a system by impersonate one of its trusted user or simply to ensure insider's anonymity. Regarding the service emulated by the honeypot studied in this chapter and its placement in the public Internet, implementing IP spoofing to ensure password guessers anonymity sounds overkill. Indeed, the packet recuperation mechanism could appear to be more or less complex for such a basic attack. Regarding DoS attacks, the Microsoft Windows RDP service can indeed be used in UDP reflection/amplification attacks with an amplification ratio of 85.9 : 1 according to Netscout. The amplified attack target victim IP address(es) and UDP port(s) traffic with UDP packets from the RDP protocol using UDP on port 3389. The forget RDP packets during the attack are consistently 1,260 bytes in length, padded with long strings of zeroes.[98] From Code 4.1, one can see that the configured RDP service for the honeypot runs over TCP and UDP, making it vulnerable to this security breach. The accessibility of port 3389 (default for RDP) for UDP has also been confirmed using Microsoft's PortQry command-line tool.[99] However, the network activity of the RDP device has been informally monitored during its whole deployment period, using the Windows Task Manager application and no high network consumption has been observed for this device. Furthermore, the status reports of Cisco Secure Endpoint for this same duration do not reveal any network anomalies. Both these actions allow to argue the little chance that the RDP connection has been implied in DDoS attacks using the breach described above. Nevertheless, the studied data set does not inventory any network metrics regarding the initiated connection with the honeypot while this would have been interesting in order to prove this assumption, which again emphasizes a limit of the data set. For both these reasons, although the ASNs resolution based on IP addresses should be considered with caution, it seems that implementing IP spoofing to contact the RDP honeypot would have turned out to not be used here.

From the last experiment described in section 6.2.2, IP address blacklisting seems to be efficient as defense mechanism against RDP brute-forcing. Indeed, 80% of the blacklisted IP addresses having contacted the honeypot did not reach it back after being whitelisted again. However, the sample allowing to make this assumption is rather small, concerning only 10 of the 134 (***i.e.*** 7.463%) of the different IP addresses of the foreign hosts having contacted the honeypot. Confirming this by implementing automatic blacklisting based on some criteria for a larger addresses sample would definitely be interesting as part of a future research, as it will be presented in section 9.2.

Still regarding future works, from the data analyzed in section 6.2, several criteria can be tested for IP address blacklisting. For instance, the connection duration has been identified as lasting around 9 minutes in average. One blacklisting policy could block

the most aggressive hosts, defined as the ones connecting to the honeypot for a period above this mean value. Its testing would be implementing by blocking them and observe if they return after being whitelisted again. One could also try to take advantage of the identified subnets, by testing if blacklisting an IP address found in one of them turns out to be more effective than not considering subnetworks.

Regarding the identified autonomous system, one can see from TABLE 6.2.3 the five autonomous systems owning the highest number of IP addresses present in the data set. AS 9829 appears in first position, being part of the Bharat Sanchar Nigam Limited (BSNL) national internet backbone (NIB) of India. This transit AS is followed by two Tier-1 ones, being AS 3257 and 174, respectively owned by Global Telecom and Technology (GTT) Inc. and Cogent Communications. Both of them are multinational Internet service providers (ISPs) based in the United States of America. The fourth and the last positions are attributed to AS 14061 and AS 212238, respectively owned by DigitalOcean and Datacamp Limited. Digging a bit more the data for AS 174, it appears that an IP range, through being part of Cogent's AS 174, is actually owned by the Heficed organization, according to MaxMind records. This company as well as DigitalOcean are both cloud service providers, offering web hosting services that have probably been used to run the password guessers with the IP addresses having target the RDP honeypot. The other companies offer wide area network services for businesses, which may includes CSPs but the trail unfortunately ends at these observations.

From time metrics analyzed in section 6.2.1, one can see from FIGURE 6.2.1a that actually, the number of performed requests increased at the end of the deployment time interval. As already mentioned, before April 4, 2022, the most aggressive IP addresses was manually blacklisted which has not be done anymore after this date. Actually, in average 4 different IP addresses has contacted the honeypot each day of its deployment, and this number does not grow after April 4, 2022. The action of blacklisting therefore really decreases the number of connections initiated with the honeypot, thus reducing the waste of network bandwidth they cause. Still in this graph, a strange gap appears in the data between March 31 and April 3, 2022. For this period of time, it has been verified that both the monitoring application and RDP honeypot were still operational, which was the case. No explanation has been found regarding the fact that no connection have been initiated with the honeypot for these days. It would be however interesting to see if this also occurs for a wider deployment time range. One last consideration regarding the data collected by the honeypot studied by day are related to March 21, 2022, on which day Belgian universities suffered DDoS attacks.[100] On that day, the RDP honeypot only experienced one connection coming from an IP address located in Mexico which lasted for approximately one hour. The DDoS attacks having taken place around 15:00, it is not impossible that the honeypot was also affected by these attacks, which would have disconnected it from the internet. Nothing seems to indicate however that these attacks were using RDP as tool to fulfill their malicious goals.

Concerning FIGURE 6.2.1b and FIGURE 6.2.1c, the rather uniform distribution of the hour and weekday activities are clues about the randomness with which hosts initiate a connection with the honeypot. Indeed, no real pattern can be extracted from these data, which can lead to think that in reality, these password guessers crawl the entire public IP range looking for victims. Once they find a machine running RDP, they try to guess the credentials with varying levels of insistence and intelligence.

Even if some limitations of the data set have been highlighted in this section, it is worthwhile to recall that the honeypot fulfilled its mission perfectly. Indeed, as explained in chapter 4, this RDP honeypot was designed to be a simple PoC, demonstrating the interest of using Cisco SecureX toolkit to monitor honeypot by collecting simple data on the black hat community. All the limitations presented above should therefore be considered rather as future improvement prospects for this device than design errors.

Elasticsearch, Logstash and Kibana (ELK) Stack Data Analysis

7.1 Data Set Review

Once again, before starting enumerating the fields of each data set record, clarifying the sources of this data set is important. While the previous chapter was presenting the data collected by the honeypot described in chapter 4, this chapter will analyze the information accumulated by the Elastic honeypot from chapter 5. As already mentioned, this honeypot is made of several devices (***cf.*** FIGURE 5.3.1). For sake of simplicity, each of these device generates its own data set[1], except for the device running Logstash and the Log Generator application which is not monitored using a SecureX workflow, as explained in section 5.4.3. Having separate data set for each device of the honey deployment will also allow to analyze the behaviors by Elastic application and not by considering the stack in general.

The data sets sources now identified, their structure can be dissected. Each record from each of the three generated data set is made of twelve fields. Five of them are exactly the same as the one presented in section 6.1 and are listed below together with the new ones.

- **Alert Timestamp:** the timestamp of the alert in RFC 3339[93] format `YYYY-MM-DDTHH:mm:ss.SSSSSSSSSZ` (***e.g.*** `2022-04-26T08:45:01.047004316Z`);
- **IP Address:** the IP address of the source initiating the RDP connection with the honeypot (***e.g.*** `192.241.221.151`);
- **AbuseIPDB Link:** the URL toward the research into AbuseIPDB database for the IP address initiating the connection with the honeypot (***e.g.*** `https://www.abuseipdb.com/check/192.241.221.151`);
- **Region:** the region where the IP address is located according to AbuseIPDB records (***e.g.*** San Francisco, California);
- **Country:** the country where the IP address is located according to AbuseIPDB records (***e.g.*** United States of America);
- **Load average:** is the average system load calculated by the `top`[91] command line utility. Three fields of this data set represent load average calculated over the different periods of
  - **1 Min**;

[1] Available on the GitHub repository of this thesis.

  - **5 Min**;
  - **15 Min**;
- **%CPU:** the percentage of CPU usage of the Elasticsearch or Kibana process as given by the `top` command line utility;
- **%MEM:** the percentage of memory usage of the Elasticsearch or Kibana process as given by the `top` command line utility;
- **Network last 2s sent:** the sending rate of a connection between the Elastic process of the honeypot node and a foreign host identified by its IP address measured using the `iftop`[90] command line utility;
- **Network last 2s received:** the receiving rate of a connection between the Elastic process of the honeypot node and a foreign host identified by its IP address measured using the `iftop` command line utility.

The reader can therefore notice that the data set is basically an updated version of the one presented in section 6.1 incremented with some host based metrics. Following the same approach than the one used in chapter 6, the following sections present the analysis of each field detailed supra.

7.2 Analysis of the Different Data

7.2.1 Timestamps

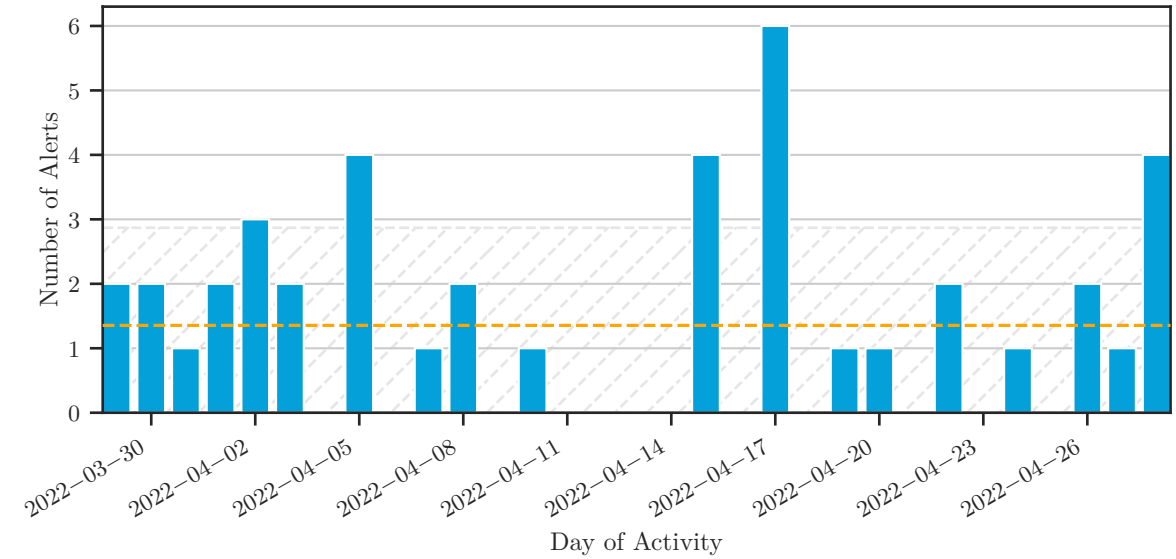
The first observation regarding timestamps data for all the three studied data set in this chapter is that no connection initiated with the Elastic honeypot has lasted more than one alert. Therefore, the study of the duration of a connection between a node of the honeypot and a foreign host would be this time of no interest. The analysis of metrics related to timestamps will therefore be limited in this chapter to their own raw content, no other data will thus be derived from them.

In the continuity of the analysis made in section 6.2.1, Figure 7.2.1-Figure 7.2.3 present graphically the timestamps information coming from the data sets for all the three monitored nodes of the Elastic honeypot. The following subsections will review the most noteworthy patterns in these figures, for each of the nodes.

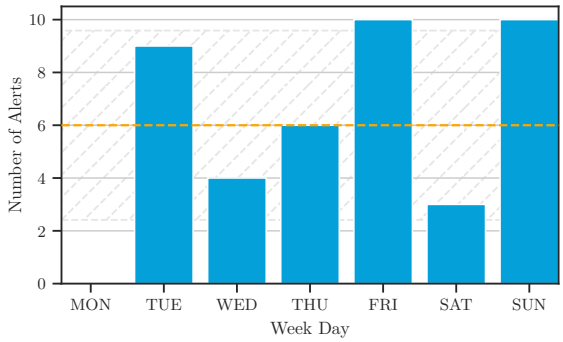
7.2.1.1 Elasticsearch Node 1

With only 42 alerts dispersed over a period of twenty-nine days, the Elasticsearch Node 1 of the honeypot deployment is the one with the lowest level of interactivity. One can see from Figure 7.2.1a or Table 7.2.1 that a connection with this host has been initiated in average 1.355 times a day. In Figure 7.2.1a, the reader can observe that no connection has been monitored on Monday during the whole honeypot deployment duration. For the same figure, it can also be seen that Friday and Sunday are the days with the highest observed activity (160% of the honeypot mean activity by week day).

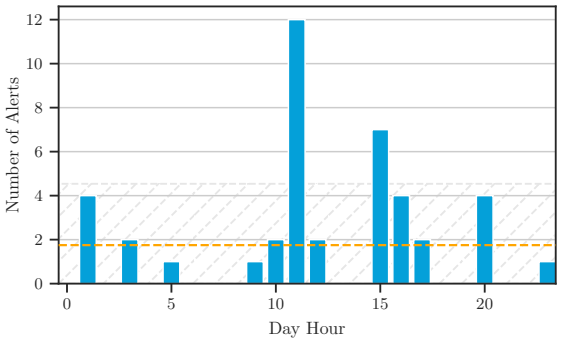
The "high" standard deviation of the timestamp hour field should attest that a pattern could possibly be observed from the graph depicted in Figure 7.2.1c. Analyzing this figure, one can indeed notice a diurnal pattern in the alert timestamps, the diurnal cycle being defined from 6:00 to 19:00. Indeed, 71.428% of the connections experienced by this node of the honeypot are within this time range, representing the majority of them.



(a) Number of connections initiated with the honeypot per its day of activity.



(b) Number of connections initiated with the honeypot per week day.



(c) Number of connections initiated with the honeypot per hour of the day.

FIGURE 7.2.1

Connections initiated with the Elasticsearch node 1 of the honeypot placed in the public Internet area presented in chapter 5. These graphs present the global timestamp metrics of the observed alerts. In each histogram, the mean value of connections by the respective metrics is represented by a dashed orange line. The gray dashed hashed area gives the standard deviation area around the mean of the data. These constant values are also given numerically in TABLE 7.2.1.

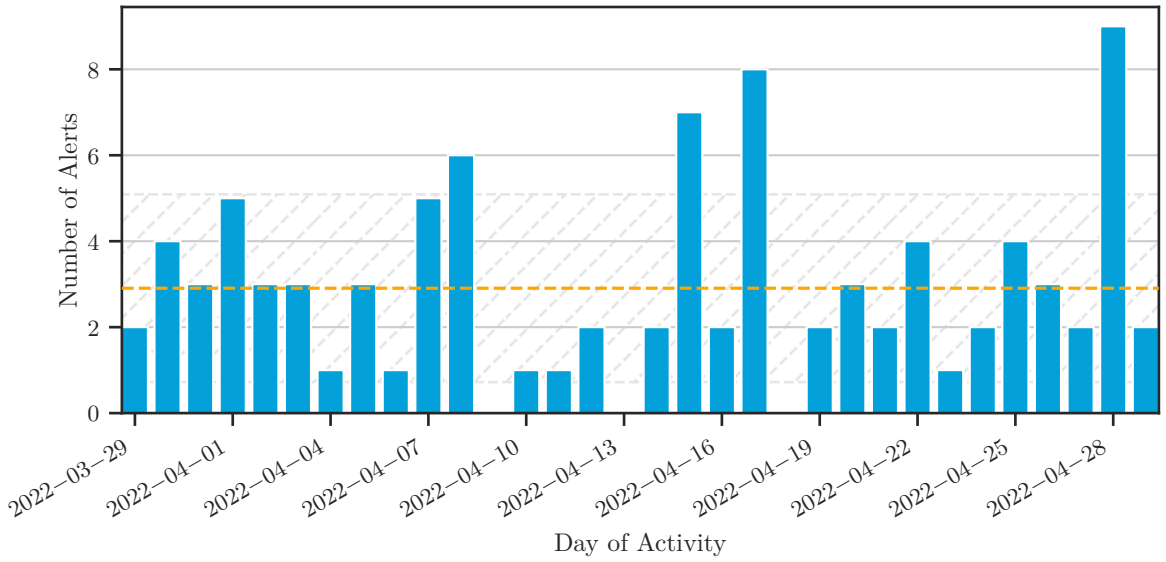
7.2.1.2 Elasticsearch Node 2

The second Elasticsearch node of the honeypot collected about twice as much data as the first one, with 93 records accumulated over thirty-one days. One can see from FIGURE 7.2.2a or TABLE 7.2.1 that this is reflected by the mean connections number by day of activity for this node, being equal to 2.906, ***i.e.*** more than the double of the one for the Elasticsearch node 1.

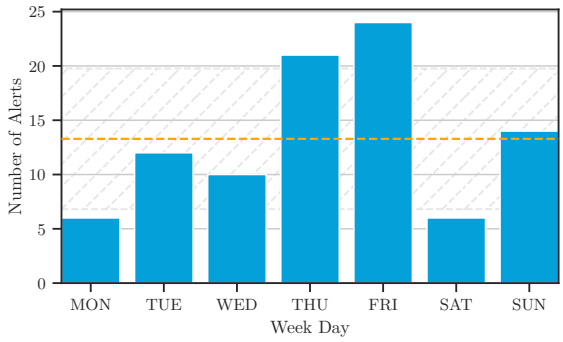
FIGURE 7.2.2b indicates that Monday remains together with Saturday the days for which the honeypot experienced the lowest interactivity while Friday is still among the days with the most connections.

Once again, the high standard deviation value for the distribution of the hours field of the timestamp data indicates that perhaps a pattern may be found inside this information. Analyzing FIGURE 7.2.2c, one can indeed notice a diurnal pattern again in the alert

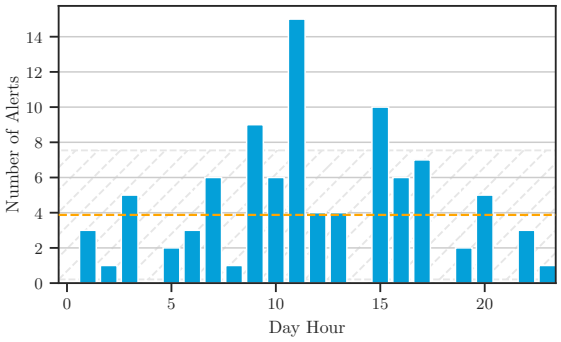
timestamps, still considering the diurnal cycle from 6:00 to 19:00. This time, 76.344% of the connections (***i.e.*** 71 connections) has been initiated during day hour with this honeypot's node.



(a) Number of connections initiated with the honeypot per day of activity.



(b) Number of connections initiated with the honeypot per week day.



(c) Number of connections initiated with the honeypot per hour.

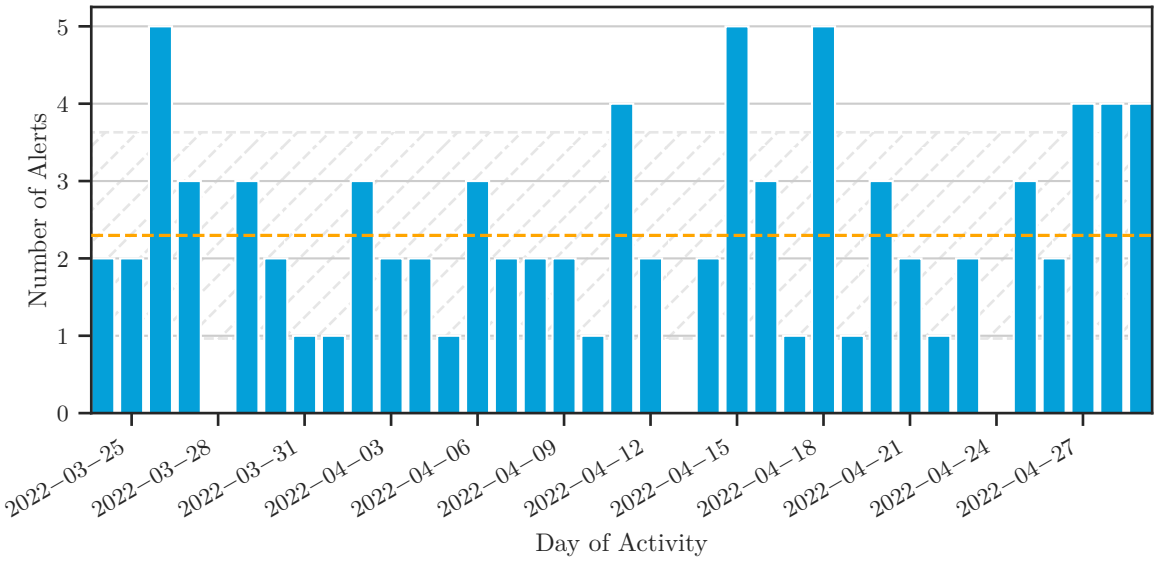
FIGURE 7.2.2

Connections initiated with the Elasticsearch node 2 of the honeypot placed in the public Internet area presented in chapter 5. These graphs present the global timestamp metrics of the observed alerts. In each histogram, the mean value of connections by the respective metrics is represented by a dashed orange line. The gray dashed hashed area gives the standard deviation area around the mean of the data. These constant values are also given numerically in TABLE 7.2.1.

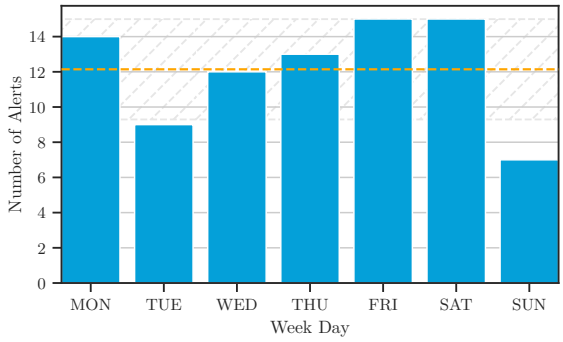
7.2.1.3 Kibana Node

The Kibana node of the Elastic honeypot was contacted 85 times during the period from March 23, 2022 to April 29, 2022 (representing a range of 36 days). The daily distribution of these alerts can be seen in FIGURE 7.2.3a. This node thus does not have the highest number of records in its data set even if it has the longest alert period of all the nodes of the Elastic honeypot (since it has been contacted by a foreign host sooner than the other nodes). This is once again reflected by the mean number of connection initiated with the honeypot during its deployment period, being of 2.297 which is lower then the one of the Elasticsearch node 2 (***i.e.*** node with the highest number of record).

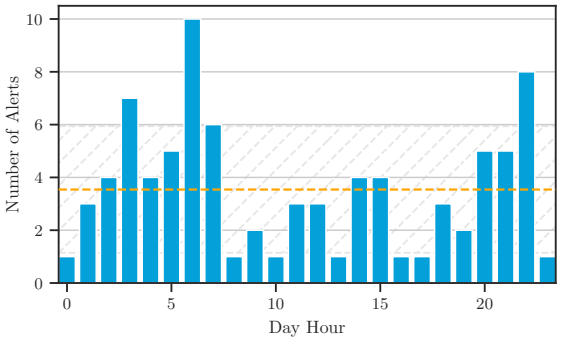
For this honeypot, no particular pattern can be observed, neither for the distribution by week day (***cf.*** FIGURE 7.2.3b) or for the one by day hour (***cf.*** FIGURE 7.2.3c) except maybe that Sunday is this time the day with the smallest number of connections.



(a) Number of connections initiated with the honeypot by its day of activity.



(b) Number of connections initiated with the honeypot by week day.



(c) Number of connections initiated with the honeypot by hours.

FIGURE 7.2.3

Connections initiated with the Kibana node of the honeypot placed in the public Internet area presented in chapter 5. These graphs present the global timestamp metrics of the observed alerts. In each histogram, the mean value of connection by the respective metrics is represented by a dashed orange line. The gray dashed hashed area gives the standard deviation area around the mean of the data. These constant values are also given numerically in TABLE 7.2.1.

7.2.2 IP Addresses

The timestamp in the data set studied, the second independent field for the records can now be analyzed, being the IP addresses of the foreign hosts having contacted this time the nodes of the Elastic honeypot as in 6. The world location of these electronic addresses has still been defined by using the platforms presented in section 6.1. FIGURE 7.2.4 and TABLE 7.2.2 give the number of IP addresses identified by geolocation for the first Elasticsearch node of the honeypot while FIGURE 7.2.5 and TABLE 7.2.3 give the ones for the second Elasticsearch node. The same metrics relative to the Kibana node of the honey deployment are given in FIGURE 7.2.6 and TABLE 7.2.4. One can see from

Elasticsearch Node 1			
	By Day of Activity	By Hour	By Week Day
Mean number of connections	1.355	1.75	6.0
Standard deviation of the number of connections	1.514	2.788	3.586
Elasticsearch Node 2			
	By Day of Activity	By Hour	By Week Day
Mean number of connections	2.906	3.875	13.286
Standard deviation of the number of connections	2.185	3.666	6.474
Kibana Node			
	By Day of Activity	By Hour	By Week Day
Mean number of connections	2.297	3.542	12.143
Standard deviation of the number of connections	1.333	2.398	2.85

TABLE 7.2.1

Mean (as represented in the FIGURE 7.2.1-FIGURE 7.2.3), variance and standard deviation of the number of connections for general day, hours or week day activity of the monitored Elastic honeypot node placed in the public Internet.

all these three figures (*resp.* tables) that most of the connection initiators IP addresses are located in the west coast of the United States. More different countries are however represented in the geolocation data of the Kibana node.

Country	Continent	Number of Connection	Proportion of Connections
United States of America	North America	42	100%

TABLE 7.2.2

Number of connections initiated with the first Elasticsearch node of the honeypot placed in the public Internet area and the proportion of the total number they represent by country, as depicted in FIGURE 7.2.4

The various subnetworks appearing in the data sets analyzed in this chapter have once again being studied. These are given in TABLE B.1.2, TABLE B.1.3 and TABLE B.1.4 respectively for the first Elasticsearch node, the second one and the Kibana node of the Elastic honeypot. An interesting observation from these tables is that, for all nodes of the honey deployment, the subnet `192.241.192.0/18` has been identified as subnet regrouping most of the IP addresses in the different data sets. Once again using the Team

FIGURE 7.2.4
World location of the IP addresses having contacted the Elasticsearch node 1 of the Elastic honeypot placed in the public Internet area.

FIGURE 7.2.5
World location of the IP addresses having contacted the Elasticsearch node 2 of the Elastic honeypot placed in the public Internet area.

Cymru tool presented in section 6.2.2 to find the AS responsible of this subnet, all the IP addresses from the data identified as part of this range appear to be owned by the same autonomous system, the one of ASN 14061, managed by the DigitalOcean Inc. company.

As in section 6.2.2, an analysis of the autonomous system responsible for the different IP addresses of the host having contacted the honeypot has been conducted for each data set studied in this chapter. FIGURE 7.2.7 shows the proportions of these ASes types for the data collected by the Elastic honey deployment. From this figure, the reader can first

Country	Continent	Number of Connection	Proportion of Connections
United States of America	North America	82	88.172%
Belgium	Europe	8	8.602%
Luxembourg	Europe	1	1.075%
China	Asia	1	1.075%
Poland	Europe	1	1.075%

TABLE 7.2.3

Number of connections initiated with the second Elasticsearch node of the honeypot placed in the public Internet area and the proportion of the total number they represent by country, as depicted in FIGURE 7.2.5

FIGURE 7.2.6

World location of the IP addresses having contacted the Kibana node of the Elastic honeypot placed in the public Internet area.

notice that the AS types are the same as the one defined in 6.2.2 but also that only transit and stub AS are responsible of IP addresses in the data set of each node of the honeypot and no tier-1 one. Another observation that can be made from these graphs is that, as in section 6.2.2, most of the identified ASs are of type transit.

As already mentioned earlier in this section, the autonomous system 14061 has been identified as the owner of most of the IP addresses of all the analyzed three data sets. To ensure information completeness, the exact proportion of IP addresses belonging to this AS is of 94.872% in the first Elasticsearch node's data set, 89.024% for the one of the second and 73.684% for the Kibana node. TABLE B.2.2-TABLE B.2.4 from the appendix B give the detailed matching made between autonomous system and the group of IP from the data set for which they own. From these tables, the general observation is that the proportion of IP addresses belonging to other ASs than the 14061 one is very negligible.

Studying the appearance of subnets in the different data sets allowed to notice sim-

Country	Continent	Number of Connection	Proportion of Connections
United States of America	North America	64	75.294%
Ireland	Europe	7	8.235%
Russia	Europe	4	4.706%
China	Asia	4	4.706%
Netherlands	Europe	2	2.353%
Hungary	Europe	1	1.176%
India	Asia	1	1.176%
Turkey	Asia	1	1.176%
Jamaica	North America	1	1.176%

TABLE 7.2.4

Number of connections initiated with the Kibana node of the honeypot placed in the public Internet area and the proportion of the total number they represent by country, as depicted in FIGURE 7.2.6

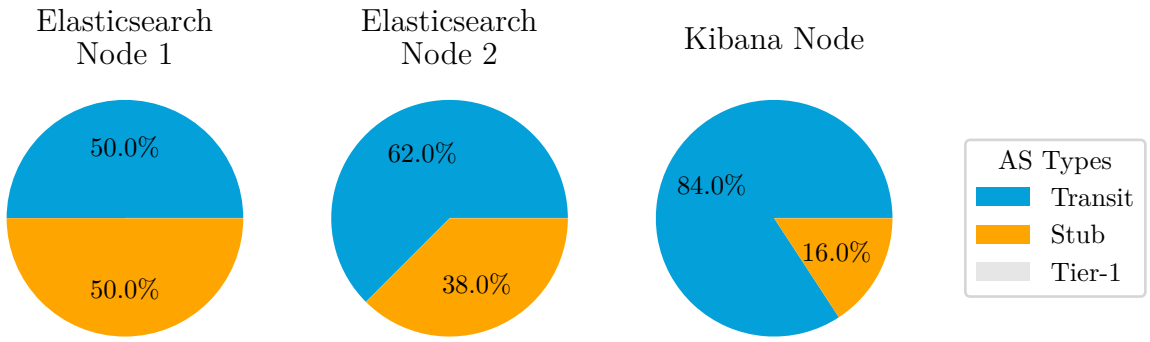


FIGURE 7.2.7

Proportion of autonomous system types (***i.e.*** stub, tier-1 or transit) resent among the AS derived as owner of the IP addresses in the data sets, from left to right, of the first Elasticsearch, second Elasticsearch and Kibana node of the Elastic honeypot.

ilarities earlier in this section. Following this observation, the overlapping of exact IP addresses field between all the collected data studied in this chapter has also been studied, in order to try to reveal more precise behavior of the foreign hosts having contacted the Elastic honeypot. TABLE 7.2.5 presents the last studied metric regarding these IP addresses, being the probability of finding one of these from a honey deployment node's data set ($\mathcal{A}$) into the one of another node ($\mathcal{B}$), denoted in this thesis $P(\mathcal{A} \cap \mathcal{B})$, following the well known intersection probability. From this table, one can see that the probability of finding an IP address from the Elasticsearch node 1 of the deployment inside the one of the second Elasticsearch node is of around 97.5%. This probability drops to 46.341% while considered the other way around, which is not surprising since the data set of the second Elasticsearch node is twice the size of the first one. However, even if a correlation appears between the IP addresses contacting the two Elasticsearch nodes of the deployment, none seem to appear between these nodes and the Kibana one.

$\mathcal{A}$ \ $\mathcal{B}$	IP Addresses of Elasticsearch Node 1	IP Addresses for Elasticsearch Node 2	IP Addresses for Kibana Node
IP Addresses of Elasticsearch Node 1	1	97.436%	7.692%
IP Addresses of Elasticsearch Node 2	46.341%	1	4.878%
IP Addresses of Kibana Node	3.947%	5.263%	1

TABLE 7.2.5

Probability of finding an IP address from one node of the Elastic honeypot data set $\mathcal{A}$ inside another one $\mathcal{B}$, ***i.e.*** intersection of the two set of IP addresses $P(\mathcal{A} \cap \mathcal{B})$.

7.2.3 System Information

Now that the general metrics of initiated connections with the honeypot has been studied, the collected system metrics can be analyzed. These metrics has been gathered using the `top` (***a.k.a.*** table of processes) Linux program, as explained in section 5.4. FIGURE 7.2.8-FIGURE 7.2.10 present the load averaged in the data sets of the monitored nodes of the Elastic honeypot device for the 1, 5 and 15 minutes preceding the connection to which they refer. Of course, analyzing these device loads would make no sense if they could not be compared to the normal system load. This normal load (***i.e.*** the load for the device not experiencing any other connections than the configured ones between the Elastic honeypot nodes) has therefore been measured on each of the monitored honeypot's devices, by simply running the `top` utility and waiting for the stabilization of the metrics. These measures are represented for the first Elasticsearch node, the second one and the Kibana node by the red dots, respectively in FIGURE 7.2.8, FIGURE 7.2.9 and FIGURE 7.2.10.

By analyzing the load average computed for the minute preceding the contact with the honeypot on FIGURE 7.2.8-FIGURE 7.2.10, one can see that a connection initiated with the Elasticsearch node 1 of the Elastic honeypot tends to decrease the device load average while it seems to be of no effect for the second Elasticsearch node. By contrast, the system load for Kibana node seems to increase most (75%) of the time when this host is contacted from outside the deployment. One can also see from these figures that the collected metrics for the Elasticsearch node 1 of the honeypot are much more dispersed than the one of the two other represented nodes.

Focusing this time only on one of these figures, FIGURE 7.2.9, the reader can see that a significant amount of outliers appear in the plots, upper outliers to be more precise. These represent the data outside 1.5 times the interquartile range (IQR) above the upper quartile (***i.e.*** $\mathcal{Q}_3 + 1.5$ IQR). However, the medians of each of the three box plots presented in this figure are close to each other. Comparing the IRQ of the box plots, the one for the average load computed for the minute preceding the alert is wider than the one for larger computation time. This translate the fact that load measures for this devices are more dispersed while computed for a small period of time and stabilize while this range is widened. This seems actually completely logical, and in fact this behavior can be observed in each of the FIGURE 7.2.8-FIGURE 7.2.10.

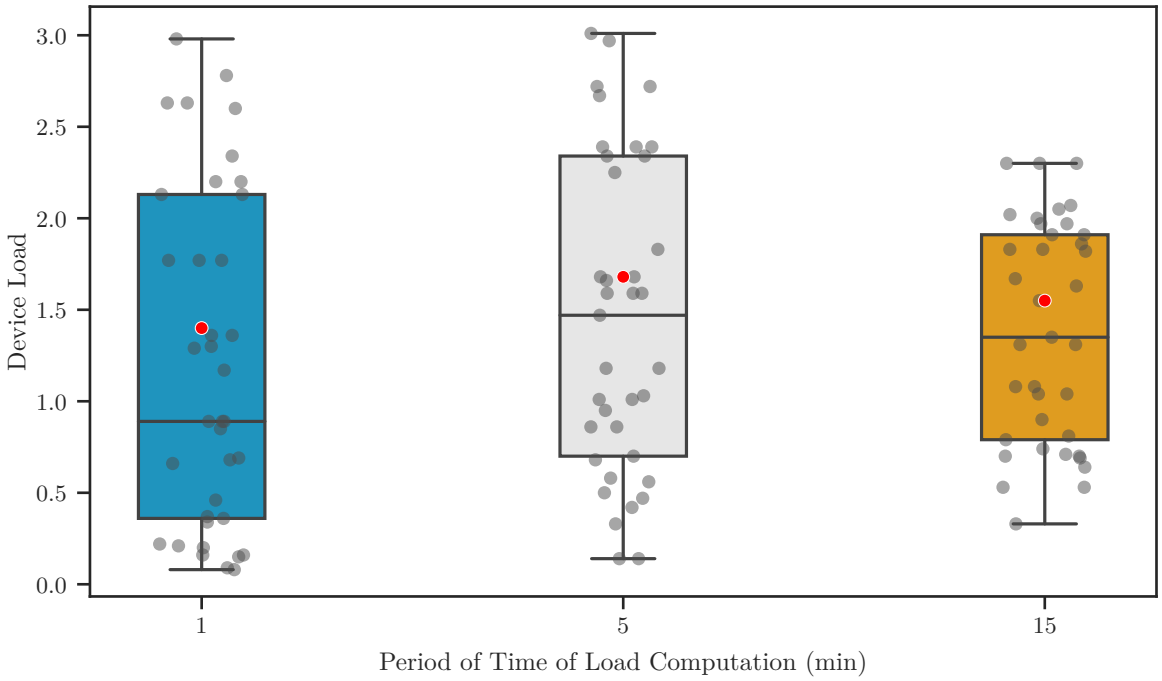


FIGURE 7.2.8

Box plot of the collected device load averages using the `top` command line utility for the Elasticsearch node 1 of the Elastic honeypot. The blue plot gives the device load evaluation for the last 1 minute. The gray one shows the same metric for the last 5 minutes. The orange box plot finally gives the device load calculated for a duration of 15 minutes. The red dots in each box plot represent the measured normal loads on the device while no connection is initiated with it.

Regarding the last metrics composing the data set presented in section 7.1 being the CPU, memory and network statistics for the different Elastic processes running inside the honeypot, these data are not presented in this section since they appeared as being of no interest after analysis. This decision will be detailed in section 7.3.

7.3 Discussion

The analysis made in section 7.2 this time seems to show that the Elastic honeypot was targeted by hybrid devices between host and port scanners, which will be referred to as scanners in this section. Indeed, the fact that no connection lasted in time and that a contact with the honeypot was initiated with a mean interval time between 8 (for the Elasticsearch node 2) and 17 hours (for the Elasticsearch node 1) shows that the foreign hosts have probably tried to connect with the honey deployment to discover which ports were open on which devices. Furthermore, the IP addresses of the hosts having initiated a connection with the first Elasticsearch node are also almost all present in the data set of the second one (***cf.*** TABLE 7.2.5). In the practical deployment presented in section 5.3, it should be noted that the four nodes of the Elastic deployment have consecutive IP addresses. Studying the overlapping IP addresses between the data sets, 97.5% of them contacted the two Elasticsearch nodes and generated an alert at the exact same

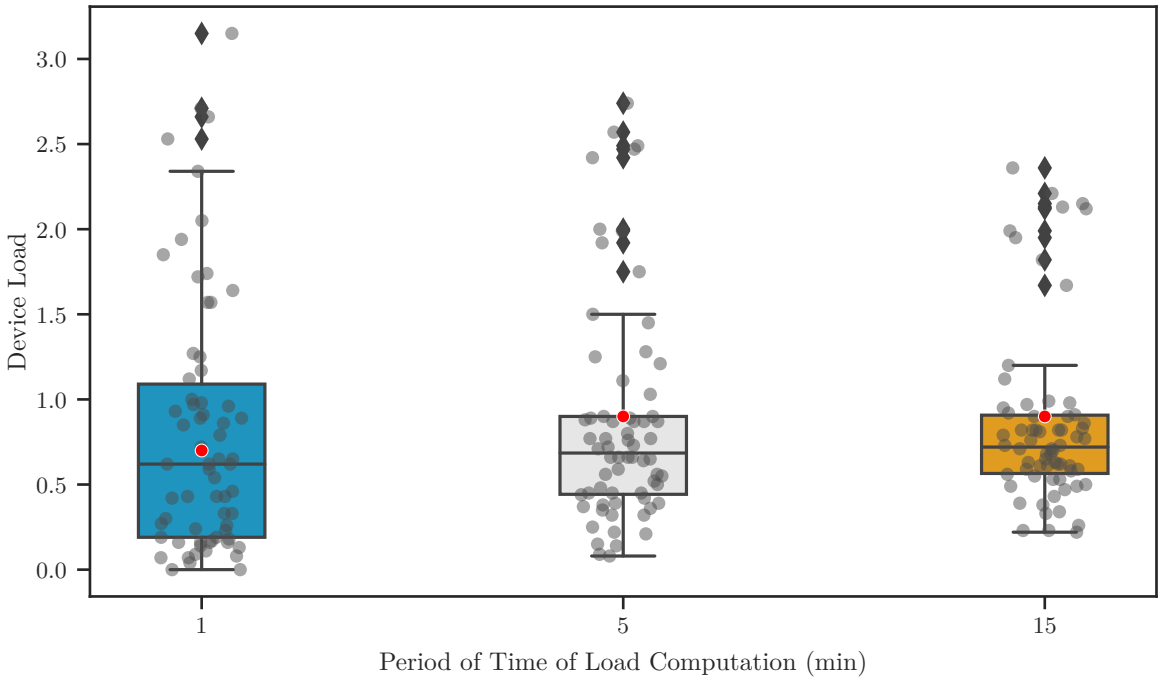


FIGURE 7.2.9

Box plot of the collected device load averages using the `top` command line utility for the Elasticsearch node 2 of the Elastic honeypot. The blue plot gives the device load evaluation for the last 1 minute. The gray one shows the same metric for the last 5 minutes. The orange box plot finally gives the device load calculated for a duration of 15 minutes. The red dots in each box plot represent the measured normal loads on the device while no connection is initiated with it.

time.[2] The remaining ones initiated connection with one node another with a delay of about 3 minutes, representing the SecureX workflow trigger delay for this deployment, as explained in section 5.4.3. This is a clear evidence that actually, most of the hosts having contacted the two Elasticsearch nodes of honeypot were in fact crawling the public IP range looking for victims. For the Kibana node, even if the proportion of overlapping IP addresses having contacted this device with the ones of the two other monitored nodes of the deployment is much lower, it is however not non-existent. Furthermore, even if the IP addresses in the data set of the Kibana node do not match perfectly the ones of the Elasticsearch nodes, one can notice that for each of these data sets, the same autonomous system (with ASN 14061) owns significant proportion of the IP addresses of the foreign hosts having contacted the honeypot. These observations allow therefore to include the Kibana in victims of the potential scanners crawling the public IP range.

The fact that the nodes of the deployment have been targeted by scanners suggests that these hosts have been implied in the first step of an attack. Indeed, even if it exists several models identifying the phases of a cyber attack such as the Cyber Kill Chain[101], the Unified Kill Chain[102] or the MITRE ATT&CK[103] frameworks among others, all reached a consensus regarding the first step of a cyber attack that they define as the reconnaissance stage. During this phase, discovering attacks are performed by insiders to

[2]Recall from section 7.2 that this timestamp represent the starting time of the workflow presented in Figure 5.4.1. The connection detection workflows for both Elasticsearch nodes of the Elastic honeypot are actually regrouped inside a unique workflow, which justify the detection of foreign hosts contact at exact same time up to the nanosecond.

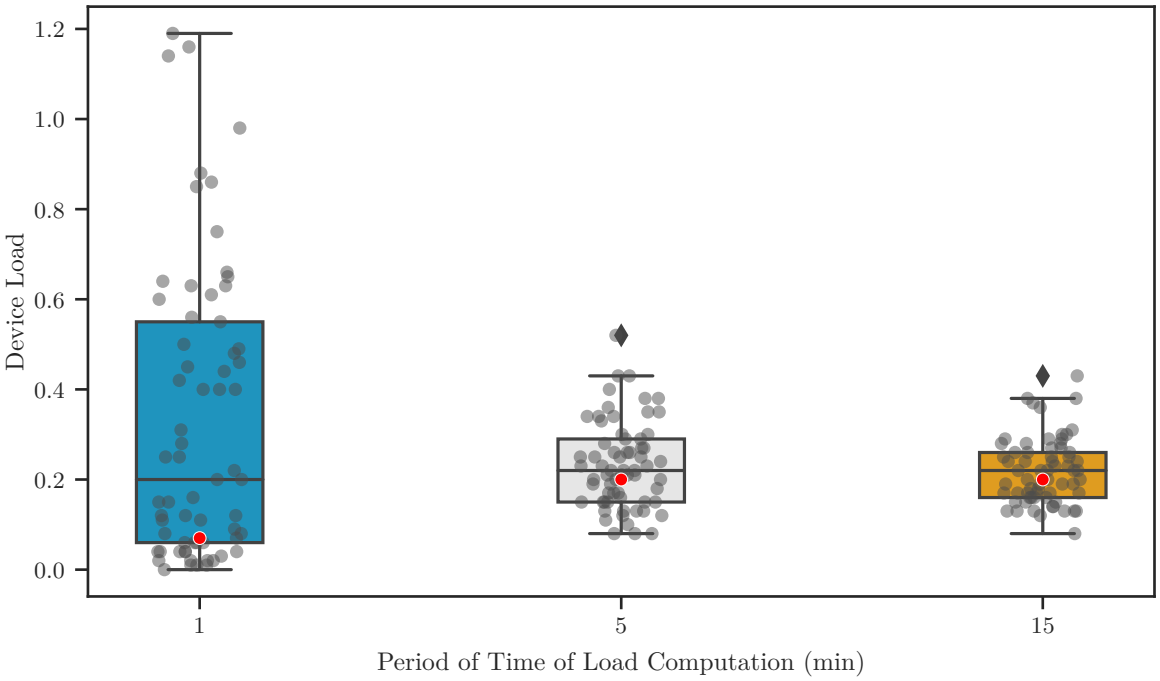


FIGURE 7.2.10

Box plot of the collected device load averages using the `top` command line utility for the Kibana node of the Elastic honeypot. The blue plot gives the device load evaluation for the last 1 minute. The gray one shows the same metric for the last 5 minutes. The orange box plot finally gives the device load calculated for a duration of 15 minutes. The red dots in each box plot represent the measured normal loads on the device while no connection is initiated with it.

get information about potential victims and identify their weaknesses. After this analysis step, the real malicious actions can take place. All the observations lead to think that the Elastic honeypot has been indexed inside the opponent's camp, however the small deployment duration does not allow to affirm that for sure. Indeed, over a larger period of time, one could have maybe observe the appearance of longer connections during which the system would have experienced one of the next stages of the attack's frameworks.

Concerning the identified autonomous systems owning the IP addresses having contacted the honeypot, the reader can notice from TABLE B.2.2-TABLE B.2.4 that AS 14061 is responsible of most of them. This AS is managed by DigitalOcean, which is a popular inexpensive American CSP for developers and SMEs as already mentioned in section 6.3. This Cloud infrastructure as a service (IaaS) platform has probably been used to host the devices running the scanners having targeted the Elastic honeypot.

From section 5.5.3, the reader may recall that special efforts had been made to try increase the interactivity with the Kibana node of the honeypot by using honeytokens. It may also be remembered from section 5.3.5.1 that the Kibana service has been configured to run on the well known HTTPS standard port 443 instead of the default one for this application (***i.e.*** port 5601). The idea behind this configuration was to define whether or not the standard ports used by well-known network protocols attracted more traffic than the others. While comparing the metrics presented in FIGURE 7.2.3 with the ones in FIGURE 7.2.1-FIGURE 7.2.2, it can be seen that these actions do not seem to bring more traffic in this particular part of the honeypot. However, from TABLE B.2.4, TABLE 7.2.4

or FIGURE 7.2.6, it can be observed that the traffic targeting Kibana comes from more different world locations. Regarding the short period of deployment duration, the fact that more connection has not been initiated with the Kibana node of the honeypot does not mean however that the honeytokens creation and standard port configuration were useless. It would be good to analyze this behavior for a longer period of time. Moreover, other techniques could also be tested to increase the visibility of the created honeytokens. Indeed, these were only placed into common internal locations of the deployment, such as the DNS of the university of Liège for the domain name of the service for instance. It would be nice to try to make these more visible, for example by forwarding the link toward the created Kibana service by email or placing it inside well known public web pages. This highlight limitations of the conducted experiments and propose possible future works around the deployment that will both be discussed in chapters 8 and 9.

As mentioned in section 7.2.3, the CPU, memory and network consumption metrics were deliberately ignored during the data analysis. These metrics were collected using `top` and `iftop` utilities, both launched remotely by the SecureX monitoring workflow of the honeypot after the detection of a connection with this deployment. Since these tools actually collect metrics in real time, these would have been of interest only for connections initiated by a foreign host lasting over time. Indeed, since in the different data sets considered in this chapter, each connection was instantaneous, the collected metrics was already outdated when the tools were launched. Nonetheless, even if the data relative to the network consumption did not provide more behavioral information on the connections, they still confirmed that these contacts did not last in time. Otherwise, these would have not been empty. For all these reasons, it has therefore been decided to abandon the treatment of these data. In the case of longer connections, one can however be persuaded of the value they will add to the collected data set. Notice that the load metrics remain notwithstanding valid, since they are for their part measured for a duration covering the instant of the initiated connections with the honeypot.

Still concerning device specific collected metrics, one can see from FIGURE 7.2.10 that a connection initiated with the Kibana node of the deployment seems to increase the load experimented by the device. This sounds perfectly natural since this service is a classical HTTPS web server and that typically this type of devices wait for a client contact before performing actions. However, from FIGURE 7.2.8, it seems that contacts with the Elasticsearch node 1 disturbs its productivity. Indeed, one can see that a connection tends to decrease the load on the device, as if the node was stopping its processing to answer to it. Nevertheless, the fact that the behaviors are different between the first and the second Elasticsearch node of the deployment cannot be precisely explain given the current form of the data set. Indeed, from FIGURE 7.2.9, it seems that a connection with the second Elasticsearch node of the deployment does not have significant impact on the device load. One clue could be that this comes from the distributed design of this application, electing a master node among the Elasticsearch cluster. Indeed, one can see from the configurations files of these nodes[78] that only the first one can be elected as master, which represents their only difference. Confirming this result over a larger data set sounds like a good idea in order to take it into account while classifying future behaviors extracted from connections.

Regarding the analyzed time metrics presented in section 6.2.1, one strange phenomenon appears in FIGURE 7.2.1b being the fact that during the 30 days data collection

period for the Elasticsearch node 1, not a single connection was initiated with this node of the honeypot on a Monday. It has been however verified for these days that the devices and monitoring tools were indeed up and running. Furthermore, metrics for the other nodes of the deployment also show that the honeypot was indeed reachable for the four Mondays of this interval of days. Given the small amount of data for this particular node, it cannot be asserted that this observation represents a real pattern within the records of the data set, which again highlights the limitations of this last. It would therefore be interesting to analyze if this pattern also appears for a longer deployment period of the honeypot before embarking on the search for potentially non-existent explanations.

Finally, it is worth to mention that the discussions about geolocation ans AS resolution based on IP addresses made in section 7.3 remain of course valid for this deployment too.

Part IV

Conclusion

Limitations

Several limitations of the current research have been highlighted throughout this Master's thesis. The paragraphs below review each of them.

First of all, chapter 6 shows that the Microsoft's remote desktop protocol honeypot presented in chapter 4 can reveal a lot of information about the attacks perpetrated against the devices enabling this functionality. However, recall from section 4.1 that this device was only supposed to be a simple proof of concept demonstrating the value of using SecureX to monitor such a device. As such, the data collected by this honeypot are very limited in the current implementation and do not allow to accurately classify the type of cyber threats having targeted the device.

Still discussing the limitations of the collected data sets, it is clear that the short deployment period does not make it possible to affirm the existence of a recurring patterns in the attacks targeting any of the two honeypots. Indeed, some parts of the deployments (***e.g.*** the Elasticsearch node 1 of the ELK honeypot) collected much less data than others, which clearly represents a limit of the data sets, specially the ones of the Elastic honeypot which are analyzed in chapter 7.2. However, even if the deployment duration was limited, the overall amount of data collected remains very satisfactory although not being perfect and allows to introduce some pattern avenues that might be identified for sure during future works.

Regarding the implementation of the honeypots and not the data they have collected, some small limitations were also identified in the presented work. One of them is the usage of certificates issued by the entrusted Elastic Certificate Tool Autogenerated CA. Even though these certificates are sufficient to implement SSL encryption inside a honeypot, this CA is not part of the trusted ones and therefore the certificate is in fact invalid. When trying to access the deployed Kibana instance from a web browser (***e.g.*** Google Chrome), this would normally trigger an alert (***e.g.*** `NET:ERR_CERT_INVALID`), requiring additional actions to be bypassed. When this can be exploited by the black hat community to attack the Elastic honeypot, it might also alert about the illegitimacy of the service. The usage of certificates created using the `elaticsearch-certutil` utility therefore weakens the scenario imagined for the deployment of the Elastic honeypot.

One can also mention as a limitation the fact that the logs generated by the Log Generator application feeding the Elastic honeypot cannot be easily recognized as being illegitimate. These logs are indeed created using common standards and do not contain any clues of their illegitimacy. While this is perfectly acceptable when these logs transit inside a dedicated space as is the case in this thesis, it could have harmful consequences

if they were found inside a production infrastructure. One should therefore necessary be careful when using this tool, which could be improved by the use of an identification mean for the generated fake logs.

Conclusion and Future work

9.1 Conclusion

In this Master's thesis honeypots have been presented that appear to be useful tools, not only to understand better the black hat community but also to actively protect the infrastructure in which they are deployed. Through the definition of powerful guidelines helping to create but also to monitor efficiently these security devices and their illustration by two deployments, it has been shown that, with limited resource and some skills only, these incredible machines can collect useful information very quickly. With the definition of an efficient monitoring strategy for each deployment, this technology also pointed out not to represent any security breach, as long as the right tools are properly configured to guard it. Speaking of the used tools, it has been shown that SecureX toolkit is a very powerful instrument to implement monitoring systems of any type, which is very convenient for the technology presented in this thesis, regarding the unlimited scope of honeypots. By helping a company with its safety assessment, this writing tried to break the risk and usage stereotypes too often associated with these machines. As such, honeypots should not be considered as high risk research systems but as an amazing active security technology helping to protect any kind of infrastructure.

9.2 Future work

As mentioned several times in this master thesis, the honeypots scope of application is only limited by the imagination of the one creating them. Since every individual is unique, it is a safe bet that each deployment of these devices will have its own particularity. Regarding in particular the honeypots created throughout this writing, they are however several future work perspectives that naturally stand out. They are listed below.

- The RDP honeypot data set could be improved by collecting device centered metrics. These will allow to classify the intensity with which the foreign hosts initiated connections with the honeypot.
- Various blacklisting policies based on the provided time and IP addresses range data analysis could be tested to define which are really efficient.
- Different methods could be tested to increase the visibility of the created honeytokens for the Kibana node of the Elastic honeypot, namely its DNS name.
- A technique could be defined to ensure the detectability and non-interference (***cf.*** properties 3.4-3.6) of the logs created by the Log Generator application while they are found into real infrastructure.

Bibliography

[1] Proximus. (Oct. 2021). "Research report cybersecurity - cybersecurity research report 2021," [Online]. Available: https://cybersecurity.proximus.be/survey-2021/research-report-cybersecurity (visited on May 21, 2022).

[2] Federal Bureau of Investigation, "Internet crime report," Internet Crime Complaint Center division of the Federal Bureau of Investigation, Tech. Rep., 2021. [Online]. Available: https://www.ic3.gov/Media/PDF/AnnualReport/2021_IC3Report.pdf (visited on Nov. 18, 2021).

[3] L. Spitzner, *Honeypots: Tracking Hackers*. USA: Addison-Wesley Longman Publishing Co., Inc., 2002, ISBN: 978-0321108951.

[4] ENISA, "Cybersecurity for smes - challenges and recommendations," European Union Agency for Cybersecurity (ENISA), Tech. Rep., Jun. 2021. DOI: 10.2824/770352. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-report-cybersecurity-for-smes (visited on Nov. 18, 2021).

[5] C. Sanders, *Intrusion Detection Honeypots: Detection Through Deception*. Applied Network Defense, 2020, ISBN: 978-1735188300. [Online]. Available: https://books.google.nl/books?id=suubzQEACAAJ.

[6] L. Spitzner, "Honeypots: Catching the insider threat," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, 2003, pp. 170–179. DOI: 10.1109/CSAC.2003.1254322.

[7] R. Palanisamy, A. A. Norman, and M. L. Mat Kiah, "Byod policy compliance: Risks and strategies in organizations," *Journal of Computer Information Systems*, vol. 62, no. 1, pp. 61–72, 2022. DOI: 10.1080/08874417.2019.1703225.

[8] S. François, *Digitalisation de la société: Quid de la cybersécurité?* Conference Presentation, 2022. [Online]. Available: https://seclists.org/honeypots/2003/q1/128 (visited on Apr. 22, 2022).

[9] C. Stoll, *The cuckoo's egg: Tracking a spy through the maze of computer espionage*. Doubleday, 1989, pp. 1–399, ISBN: 978-0-3852-4946-1.

[10] C. Kelly, N. Pitropakis, A. Mylonas, S. McKeown, and W. J. Buchanan, "A comparative analysis of honeypots on different cloud platforms," *Sensors*, vol. 21, no. 7, 2021, ISSN: 1424-8220. DOI: 10.3390/s21072433. [Online]. Available: https://www.mdpi.com/1424-8220/21/7/2433.

[11] Y. Sun, Z. Tian, M. Li, S. Su, X. Du, and M. Guizani, "Honeypot identification in softwarized industrial cyber–physical systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5542–5551, 2021. DOI: 10.1109/TII.2020.3044576.

[12] M. Baykara and R. Das, "A novel honeypot based security approach for real-time intrusion detection and prevention systems," *Journal of Information Security and Applications*, vol. 41, pp. 103–116, 2018, ISSN: 2214-2126. DOI: 10.1016/j.jisa.2018.06.004.

[13] P. Lackner, "How to mock a bear: Honeypot, honeynet, honeywall & honeytoken: A survey," in *Proceedings of the 23rd International Conference on Enterprise Information Systems*, INSTICC, vol. 2, SciTePress, 2021, pp. 181–188, ISBN: 978-989-758-509-8. DOI: 10.5220/0010400001810188.

[14] U. J. C. Pramodya, K. T. Y. U. De Silva Wijesiriwardhana, K. T. D. Dharmakeerthi, E. A. K. V. Athukorala, A. N. Senarathne, and D. Tharindu, "Agenthunt: Honeypot and ids based network monitoring device to secure home networks," in *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3*, K. Arai, Ed., Cham: Springer International Publishing, 2022, pp. 194–207, ISBN: 978-3-030-89912-7.

[15] X. Wei and D. Yang, "Study on active defense of honeypot-based industrial control network," in *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, 2021, pp. 2019–2022. DOI: 10.1109/HPCC-DSS-SmartCity-DependSys53884.2021.00301.

[16] K. D. Singh, "Securing of cloud infrastructure using enterprise honeypot," in *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, 2021, pp. 1388–1393. DOI: 10.1109/ICAC3N53548.2021.9725389.

[17] J. Jafarian and A. Niakanlahiji, "Delivering honeypots as a service," Jan. 2020. DOI: 10.24251/HICSS.2020.227.

[18] S. Lee, A. Abdullah, N. Jhanjhi, and S. Kok, "Classification of botnet attacks in iot smart factory using honeypot combined with machine learning," *PeerJ Computer Science*, vol. 7, e350, 2021. DOI: 10.7717/peerj-cs.350.

[19] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac, "A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2351–2383, 2021.

[20] E. D. Saputro, Y. Purwanto, and M. F. Ruriawan, "Medium interaction honeypot infrastructure on the internet of things," in *2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, IEEE, 2021, pp. 98–102, ISBN: 978-1-7281-9448-6.

[21] CyberSRC. (Jul. 2021). "Honeypot as service - CyberSRC," [Online]. Available: https://cybersrcc.com/cybersrc-honeypot-service/ (visited on Jun. 5, 2022).

[22] Blumira. (Mar. 2021). "Virtual honeypot software | honeypot software - Blumira," [Online]. Available: https://www.blumira.com/product/honeypots/ (visited on Jun. 5, 2022).

[23] J. Cândido, M. Aniche, and A. van Deursen, "Log-based software monitoring: A systematic mapping study," *PeerJ Computer Science*, vol. 7, e489, 2021. [Online]. Available: http://arxiv.org/abs/1912.05878.

[24] E. Mendes and F. Petrillo, "Towards logging noisiness theory: Quality aspects to characterize unwanted log entries," *CoRR*, vol. abs/2106.03018, 2021. arXiv: 2106.03018. [Online]. Available: https://arxiv.org/abs/2106.03018.

[25] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry.," in *ICSE 2014*, Jul. 2014. DOI: 10.1145/2591062.2591175. [Online]. Available: https://www.microsoft.com/en-us/research/publication/developers-log-empirical-study-logging-practices-industry-2/.

[26] B. Chen, "Improving the software logging practices in devops," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 194–197. DOI: 10.1109/ICSE-Companion.2019.00080.

[27] R. A. Grimes, *Honeypots for Windows*, 1st ed. Apress, 2005, pp. 1–424, ISBN: 978-1-59059-335-6.

[28] B. Li, Y. Xiao, Y. Shi, Q. Kong, Y. Wu, and H. Bao, "Anti-honeypot enabled optimal attack strategy for industrial cyber-physical systems," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 250–261, 2020. DOI: 10.1109/OJCS.2020.3030825.

[29] O. Hayatle, A. Youssef, and H. Otrok, "Dempster-shafer evidence combining for (anti)-honeypot technologies," *Information Security Journal: A Global Perspective*, vol. 21, no. 6, pp. 306–316, 2012, ISSN: 1939-3555. DOI: 10.1080/19393555.2012.738375.

[30] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Towards systematic honeytoken fingerprinting," in *13th International Conference on Security of Information and Networks*, Association for Computing Machinery, Nov. 2020, pp. 1–5, ISBN: 9781450387514. DOI: 10.1145/3433174.3433599.

[31] J. Uitto, S. Rauti, S. Laurén, and V. Leppänen, "A survey on anti-honeypot and anti-introspection methods," in *Recent Advances in Information Systems and Technologies*, vol. 570, 2017, ISBN: 978-3-319-56537-8. DOI: 10.1007/978-3-319-56538-5_13.

[32] R. Joshi and A. Sardana, *Honeypots: a new paradigm to information security*. CRC Press, 2011.

[33] N. Kambow and L. K. Passi, "Honeypots: The need of network security," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 5, pp. 6098–6101, 2014.

[34] L. Spitzner, "Honeypots: Catching the insider threat," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, 2003, pp. 170–179. DOI: 10.1109/CSAC.2003.1254322.

[35] ——, "Honeypots: Catching the insider threat," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, IEEE, 2003, pp. 170–179.

[36] G. Bell and A. Elang, "Network malware laboratory based on honeypots technologies," *Journal of Cybersecurity Research (JCR)*, vol. 3, pp. 1–12, Dec. 2018. DOI: 10.19030/jcr.v3i1.10226.

[37] ENISA, "General report," European Union Agency for Cybersecurity (ENISA), Tech. Rep., 2012. DOI: 10.2824/16829. [Online]. Available: https://www.enisa.europa.eu/publications/corporate/general-report-2012 (visited on Nov. 18, 2021).

[38] Björn Bengtson. (Dec. 2021). "Bap - http basic authentication honeypot," [Online]. Available: https://github.com/bjeborn/basic-auth-pot (visited on Dec. 20, 2021).

[39] Cymmetria. (Dec. 2021). "Cisco asa honeypot," [Online]. Available: https://github.com/cymmetria/ciscoasa_honeypot (visited on Dec. 20, 2021).

[40] Niels Provos. (Dec. 2021). "Developments of the honeyd virtual honeypot," [Online]. Available: http://www.honeyd.org/index.php (visited on Dec. 20, 2021).

[41] NETSEC. (Dec. 2021). "Specter - intrusion detection system," [Online]. Available: http://www.specter.com/ (visited on Dec. 20, 2021).

[42] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th. Wiley Publishing, 2012, ISBN: 1118063333.

[43] Cowrie. (Dec. 2021). "Cowrie," [Online]. Available: https://www.cowrie.org/ (visited on Dec. 20, 2021).

[44] D. Salmon. (Dec. 2021). "Miniprint," [Online]. Available: https://github.com/sa7mon/miniprint (visited on Dec. 20, 2021).

[45] Be the root. (Dec. 2021). "Sticky elephant," [Online]. Available: https://github.com/betheroot/sticky_elephant (visited on Dec. 20, 2021).

[46] R. Steenson and C. Seifert. (Jul. 2021). "Capture hpc," [Online]. Available: https://www.honeynet.org/projects/old/capture-hpc/ (visited on Dec. 1, 2021).

[47] The Honeynet Project. (Dec. 2021). "Sebek homepage," [Online]. Available: https://honeynet.onofri.org/tools/sebek/ (visited on Dec. 27, 2021).

[48] R. Vermeulen. (Dec. 2021). "Argos - an emulator for capturing zero-day attacks," [Online]. Available: https://www.few.vu.nl/argos/ (visited on Dec. 27, 2021).

[49] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, *Computer Security Incident Handling Guide*, en, 61. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, Aug. 2012, vol. 800, pp. 1–147. [Online]. Available: 10.6028/NIST.SP.800-61r2.

[50] Prometheus Authors. (Feb. 2022). "Prometheus - monitoring system & time series database," [Online]. Available: https://prometheus.io/ (visited on Feb. 5, 2022).

[51] I. Mokube and M. Adams, "Honeypots: Concepts, approaches, and challenges," in *Proceedings of the 45th annual southeast regional conference*, 2007, pp. 321–326.

[52] Wikipedia Community. (May 2022). "The boy who cried wolf," [Online]. Available: https://en.wikipedia.org/wiki/The_Boy_Who_Cried_Wolf (visited on May 3, 2022).

[53] J. Sullivan, *Extended Detection and Response (XDR) For Dummies, Cisco Special Edition*, en. John Wiley & Sons, Inc., 2022, pp. 1–46, ISBN: 978-1-119- 84557-7.

[54] Osquery. (Dec. 2021). "Osquery | easily ask questions about your linux, windows, and macos infrastructure," [Online]. Available: https://osquery.io/ (visited on Dec. 27, 2021).

[55] Cisco Systems, Inc. (Dec. 2021). "Orbital help," [Online]. Available: https://orbital.amp.cisco.com/help/ (visited on Dec. 21, 2021).

[56] S. Widup, A. Pinto, D. Hylender, G. Bassett, and p. langlois philippe, "Data breach investigations report," Verizon, Tech. Rep., May 2021. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/ (visited on Apr. 18, 2022).

[57] CrowdStrike, "Global threat report," CrowdStrike Holdings, Inc., Tech. Rep., 2022. [Online]. Available: https://www.crowdstrike.com/global-threat-report/ (visited on Apr. 18, 2022).

[58] J. A. Sava. (Feb. 2022). "Total spending on global information security market by segment 2017-2021," [Online]. Available: https://www.statista.com/statistics/790834/spending-global-security-technology-and-services-market-by-segment/#:~:text=Spending%5C%20on%5C%20security%5C%20services%5C%20in,reach%5C%2072.5%5C%20billion%5C%20U.S.%5C%20dollars. (visited on May 3, 2022).

[59] A. Paes de Barros, *Res: Protocol anomaly detection ids - honeypots*, Email communication, 2003. [Online]. Available: https://seclists.org/honeypots/2003/q1/128 (visited on Apr. 22, 2022).

[60] L. Spitzner, "Honeytokens: The other honeypot," Jul. 2003. [Online]. Available: https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=74450cf5-2f11-48c5-8d92-4687f5978988&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments (visited on Apr. 16, 2022).

[61] B. Bowen, S. Hershkop, A. Keromytis, and S. Stolfo, "Baiting inside attackers using decoy documents," in *Security and Privacy in Communication Networks*, Berlin, Heidelberg: Springer, Sep. 2009, pp. 51–70, ISBN: 978-3-642-05284-2. DOI: 10.1007/978-3-642-05284-2_4.

[62] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 1st ed. Chapman & Hall/CRC, Aug. 2007, ISBN: 978-1-58488-551-1. DOI: 10.1201/9781420010756.

[63] IBM Security, "Cost of a data breach report," IBM, Tech. Rep., 2012. [Online]. Available: https://www.ibm.com/security/data-breach (visited on Apr. 18, 2022).

[64] StatCounter. (Sep. 2021). "Desktop operating system market share worldwide," [Online]. Available: https://gs.statcounter.com/os-market-share/desktop/worldwide (visited on May 3, 2022).

[65] Security Space. (Mar. 2022). "Web server survey," [Online]. Available: https://secure1.securityspace.com/s_survey/data/202202/index.html (visited on Apr. 9, 2022).

[66] Cisco Systems, Inc. (Mar. 2022). "Secure endpoint user guide," [Online]. Available: https://docs.amp.cisco.com/en/SecureEndpoint/Secure%5C%20Endpoint%5C%20User%5C%20Guide.pdf (visited on Apr. 25, 2022).

[67] ——, (Jul. 2020). "Getting started — cisco securex integration workflows," [Online]. Available: https://ciscosecurity-sx-00-integration-workflows.readthedocs-hosted.com/en/latest/orchestration/getting_started.html (visited on Apr. 25, 2022).

[68] A. W. Conklin and G. White, *Principles of Computer Security: CompTIA Security+ and Beyond*, 5th ed. McGraw-Hill Education, Jun. 2018, ISBN: 978-1-260-02601-6.

[69] Wikipedia's Community. (Mar. 2022). "Black hat (computer security)," [Online]. Available: https://en.wikipedia.org/wiki/Black_hat_(computer_security) (visited on Apr. 11, 2022).

[70] D. Danchev. (Jun. 2010). "Study finds the average price for renting a botnet," [Online]. Available: https://www.zdnet.com/article/study-finds-the-average-price-for-renting-a-botnet/ (visited on May 3, 2022).

[71] Telecommunication Standardization Sector, "T.128: Multipoint application sharing," International Telecommunication Union, Standard, Jun. 2008. [Online]. Available: https://www.itu.int/rec/T-REC-T.128 (visited on Apr. 9, 2022).

[72] ——, "T.122: Multipoint communication service - service definition," International Telecommunication Union, Standard, Feb. 1998. [Online]. Available: https://www.itu.int/rec/T-REC-T.122 (visited on Apr. 9, 2022).

[73] ——, "T.125: Multipoint communication service protocol specification," International Telecommunication Union, Standard, Feb. 1998. [Online]. Available: https://www.itu.int/rec/T-REC-T.125 (visited on Apr. 9, 2022).

[74] Y. Pouffary and A. Young, "Iso transport service on top of tcp (itot)," RFC Editor, RFC 2126, Mar. 1997. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2126 (visited on Apr. 9, 2022).

[75] Telecommunication Standardization Sector, "X.224 : Information technology - open systems interconnection - protocol for providing the connection-mode transport service," International Telecommunication Union, Standard, Nov. 1995. [Online]. Available: https://www.itu.int/rec/T-REC-X.224 (visited on Apr. 9, 2022).

[76] S. Reiner. (Jul. 2020). "Explain like i'm 5: Remote desktop protocol (rdp)," [Online]. Available: https://www.cyberark.com/resources/threat-research-blog/explain-like-i-m-5-remote-desktop-protocol-rdp (visited on Apr. 9, 2022).

[77] Thinkst Applied Research. (Mar. 2022). "Opencanary," [Online]. Available: https://github.com/thinkst/opencanary (visited on Apr. 8, 2022).

[78] G. Deflandre. (Sep. 2022). "Thesis honeypot data analizer," [Online]. Available: https://github.com/Guilian-Deflandre/honeypot-data-analizer (visited on Jun. 8, 2022).

[79] Elasticsearch B.V. (May 2022). "Our story | elastic," [Online]. Available: https://www.elastic.co/about/history-of-elasticsearch (visited on May 4, 2022).

[80] D. Horovits. (Jun. 2020). "The complete guide to the elk stack," [Online]. Available: https://logz.io/learn/complete-guide-elk-stack/ (visited on May 2, 2022).

[81] The Apache Software Foundation. (Mar. 2022). "Welcome to apache lucene," [Online]. Available: https://lucene.apache.org/ (visited on May 3, 2022).

[82] Elasticsearch B.V. (May 2022). "Welcome to elastic docs," [Online]. Available: https://www.elastic.co/guide/index.html (visited on May 3, 2022).

[83] C. Gormley and Z. Tong, *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*, 1st. O'Reilly Media, Inc., 2015, ISBN: 978-1-449-35854-9.

[84] Elasticsearch B.V. (May 2022). "Elastic," [Online]. Available: https://github.com/elastic (visited on May 3, 2022).

[85] G. Deflandre. (Sep. 2022). "Log generator," [Online]. Available: https://github.com/Guilian-Deflandre/thesis-log-generator (visited on Jun. 8, 2022).

[86] D. Faraglia. (May 2022). "Faker - pypi," [Online]. Available: https://pypi.org/project/Faker/ (visited on May 30, 2022).

[87] A. Robbins and D. MacKenzie. (2010). "Iftop(8) - linux man page," [Online]. Available: https://linux.die.net/man/1/id (visited on May 11, 2022).

[88] ISO Central Secretary, "Date and time - representations for information interchange - part 1: Basic rules," en, International Organization for Standardization, Geneva, CH, Standard ISO 8601-1:2019, 2019. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:8601:-1:ed-1:v1:en.

[89] Joda.org. (Sep. 2022). "Datetimeformat (joda-time 2.10.14 api)," [Online]. Available: https://www.joda.org/joda-time/apidocs/org/joda/time/format/DateTimeFormat.html (visited on May 3, 2022).

[90] P. Warren. (Nov. 2010). "Iftop(8) - linux man page," [Online]. Available: https://linux.die.net/man/8/iftop (visited on Apr. 11, 2022).

[91] J. C. Warner and C. Small. (Aug. 2021). "Top(1) - linux man page," [Online]. Available: https://linux.die.net/man/1/top (visited on Apr. 11, 2022).

[92] A. Barfar and S. Mohammadi, "Honeypots: Intrusion deception," *The Information Systems Security Association (ISSA) Journal*, pp. 28–31, Jan. 2007.

[93] C. Newman and G. Klyne, "Date and Time on the Internet: Timestamps," RFC Editor, RFC 3339, Jul. 2002, 18 pp. [Online]. Available: https://www.rfc-editor.org/info/rfc3339.

[94] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Boston, MA: Pearson Education Limited, 2017, ISBN: 978-1-292-15359-9.

[95] Team Cymru. (May 2022). "Ip to asn mapping service - team cymru," [Online]. Available: https://team-cymru.com/community-services/ip-asn-mapping/ (visited on May 28, 2022).

[96] CAIDA. (May 2022). "Index of /datasets," [Online]. Available: https://public-data.caida.org/datasets/ (visited on May 28, 2022).

[97] Center for Applied Internet Data Analysis. (Jun. 2022). "State of ip spoofing," [Online]. Available: https://spoofer.caida.org/summary.php (visited on Jun. 2, 2022).

[98] R. Dobbins and S. Bjarnason. (Jan. 2021). "Microsoft remote desktop protocol (rdp) reflection/amplification ddos attack mitigation recommendations," [Online]. Available: https://www.netscout.com/blog/asert/microsoft-remote-desktop-protocol-rdp-reflectionamplification (visited on May 21, 2022).

[99] Microsoft. (Dec. 2021). "Using the portqry command-line tool," [Online]. Available: https://docs.microsoft.com/en-us/troubleshoot/windows-server/networking/portqry-command-line-port-scanner-v2 (visited on May 11, 2022).

[100] C. Hutin, "Les universités francophones victimes d'une cyberattaque," *Le Soir*, Mar. 2022. [Online]. Available: https://www.lesoir.be/431382/article/2022-03-21/les-universites-francophones-victimes-dune-cyberattaque (visited on Jun. 2, 2022).

[101] E. M. Hutchins, M. J. Cloppert, R. M. Amin, *et al.*, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," 1, vol. 1, Mar. 2011, p. 80. [Online]. Available: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf (visited on Jun. 1, 2022).

[102] P. Pols, "The unified kill chain," Tech. Rep., 2017. [Online]. Available: https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf (visited on Jun. 1, 2022).

[103] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," Tech. Rep., 2018. [Online]. Available: https://attack.mitre.org/docs/ATTACK%5C_Design%5C_and%5C_Philosophy%5C_March%5C_2020.pdf (visited on Jun. 1, 2022).

[104] D. Storey, "Catching flies with honey tokens," *Network Security*, no. 11, pp. 15–18, Nov. 2009.

[105] R. P. Jover, "Security analysis of SMS as a second factor of authentication," *Communications of the ACM*, vol. 63, no. 12, pp. 46–52, 2020.

[106] R. Karthikeyan, D. T. Geetha, S. Vijayalakshmi, and R. Sumitha, "Honeypots for network security," *International journal for Research & Development in Technology*, vol. 7, no. 2, pp. 62–66, 2017.

[107] Amazon Web Services, Inc. (May 2022). "The ELK stack," [Online]. Available: https://aws.amazon.com/opensearch-service/the-elk-stack/ (visited on May 3, 2022).

[108] Oxford University Press. (Dec. 2021). "Interactivity noun - definition," [Online]. Available: https://www.oxfordlearnersdictionaries.com/definition/english/interactivity?q=interactivity (visited on Dec. 18, 2021).

[109] Elasticsearch B.V. (Jan. 2022). "Free and open search: The creators of elasticsearch, elk & kibana | elastic," [Online]. Available: https://www.elastic.co/ (visited on Jan. 13, 2022).

[110] I. Goddijn and Cyber Risk Analytics Research Team, "2020 Year End Report - Data Breach QuickView," Risk Based Security, Tech. Rep., Jan. 2021.

[111] J. White, "Creating personally identifiable honeytokens," in *Innovations and Advances in Computer Sciences and Engineering*, T. Sobh, Ed., Dordrecht: Springer Netherlands, 2010, pp. 227–232, ISBN: 978-90-481-3658-2.

[112] E. Vasilomanolakis, S. Karuppayah, P. Kikiras, and M. Mühlhäuser, "A honeypot-driven cyber incident monitor: Lessons learned and steps ahead," in *8th International Conference on Security of Information and Networks*, Sochi, Russia, Sep. 2015. DOI: 10.1145/2799979.2799999.