



Assignment of master's thesis

Title:	Malware Persistence Techniques and its Detection
Student:	Bc. Martin Mandík
Supervisor:	Ing. Simona Fornůsek, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2024/2025

Instructions

Research and examine the various techniques employed by malware for persistence on a compromised Windows OS system. Provide an overview and explain the significance of malware persistence in the context of cybersecurity.

In the practical part of the thesis, develop a simulated environment for detection of malware persistence. Select suitable monitoring and detection tools and mechanisms and create a set of rules for automated detection and alerting. Examine options of automated acquisition of forensic material related to an alert. Test and assess the effectiveness and efficiency of the developed detection mechanisms.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Master's thesis

Malware Persistence Techniques and its Detection

Bc. Martin Mandík

Department of Information Security
Supervisor: Ing. Simona Fornůšek, Ph.D.

May 9, 2024

Acknowledgements

I would first like to thank my supervisor, Ing. Simona Fornůsek, Ph.D., for providing guidance during the process of writing the thesis. Her willingness to guide me and consult with me at any time really made it easier for me to complete the work. Her expertise was truly helpful to me and her clear answers to my questions saved me a lot of time.

I would also like to thank my family that has supported me throughout the time of my studies, providing the support and motivation I needed and creating a great, comfortable environment for work.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 9, 2024

Czech Technical University in Prague
Faculty of Information Technology
© 2024 Martin Mandík. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Mandík, Martin. *Malware Persistence Techniques and its Detection*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstrakt

Tato práce se zabývá tématem perzistence malwaru a v teoretické části detailně analyzuje způsoby a techniky používané ve škodlivých programech pro zajištění opakovaného spuštění škodlivého kódu. Jednotlivé techniky jsou klasifikovány dle matice *MITRE ATT&CK*. Dále práce navazuje implementací detekčního řešení, které obsahuje sadu pravidel určených pro odhalení analyzovaných způsobů persistence. Laboratorní prostředí je vytvořeno v cloudové technologii *Azure* za použití nástroje *Splunk* pro log management. Práce se věnuje také tématu automatické akvizice artefaktů, přičemž je nasazen nástroj *Google Rapid Response*, který se automaticky v integraci s detekční platformou stará o akvizici zajímavého materiálu pro analýzu.

Klíčová slova perzistence malwaru, MITRE ATT&CK, Splunk, Google Rapid Response, sběr logů, SIEM, akvizice forenzních artefaktů

Abstract

This thesis deals with the topic of malware persistence, focusing on what techniques are used by these pieces of malicious software to launch repeatedly on target machines, and investigating them in detail in the theoretical part. The techniques are classified in alignment with the *MITRE ATT&CK* matrix. Based on this research, a solution including a set of rules for detecting selected persistence techniques is created in an *Azure* cloud laboratory environment utilizing the *Splunk* log management tool. In addition, the topic of automatic artifact acquisition is explored, while deploying the *Google Rapid Response* tool to collect interesting files automatically in coordination with the detection platform.

Keywords malware persistence, MITRE ATT&CK, Splunk, Google Rapid Response, log management, SIEM, forensic artifact acquisition

Contents

Introduction	1
1 Theoretical Background	3
1.1 Malware	3
1.1.1 Malware Families	3
1.1.2 Objectives of Malware and Importance of Persistence	4
1.2 Persistence	4
1.2.1 Persistence Techniques Classification	5
1.2.1.1 MITRE ATT&CK Framework	5
1.2.1.2 Other Classification Alternatives	6
1.3 Forensic Artifacts	7
1.3.1 Windows Event Logs	8
1.3.2 Windows Registry	9
2 Exploring Persistence Techniques	11
2.1 Identifying Popular Persistence Techniques	11
2.1.1 Frequently Referred Persistence Techniques	12
2.1.2 Techniques Within Popular Malware Families	12
2.1.2.1 Examining Selected Families	15
2.1.3 Observations	15
2.2 Detailed Overview of Persistence Techniques	16
2.2.1 Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)	16
2.2.1.1 Principle	17
2.2.1.2 Examples	18
2.2.1.3 Detection	21
2.2.1.4 Artifacts	22
2.2.2 Scheduled Task/Job: Scheduled Task (T1053.005)	22
2.2.2.1 Principle	22

2.2.2.2	Examples	23
2.2.2.3	Detection	24
2.2.2.4	Artifacts	24
2.2.3	Windows Management Instrumentation (WMI) Event Subscription (T1546.003)	25
2.2.3.1	Principle	26
2.2.3.2	Examples	28
2.2.3.3	Detection	29
2.2.3.4	Artifacts	29
2.2.4	BITS Jobs (T1197)	30
2.2.4.1	Principle	30
2.2.4.2	Examples	31
2.2.4.3	Detection	32
2.2.4.4	Artifacts	32
2.2.5	Windows Service (T1543.003)	32
2.2.5.1	Principle	32
2.2.5.2	Examples	33
2.2.5.3	Detection	34
2.2.5.4	Artifacts	34
3	Laboratory Environment	35
3.1	Cloud Environment Design	35
3.1.1	Log Management	35
3.1.1.1	Splunk Components	36
3.1.1.2	Architecture of Splunk in the Lab	37
3.1.2	Remote Artifact Collection	37
3.1.3	Windows Endpoint Machine	38
3.2	Setting up Azure Environment	38
3.2.1	Splunk Enterprise Instance	39
3.2.2	Google Rapid Response Instance	40
3.2.3	Windows Endpoint Machine	40
3.2.4	Setting up Log Flow	41
4	Implementing Detection Mechanisms	43
4.1	Splunk Alerts	43
4.2	Utilization of GRR	44
4.2.1	GRR and Splunk Integration	44
4.2.2	Acquiring Files through GRR API	47
4.3	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)	48
4.3.1	Additional Endpoint Configuration	48
4.3.2	Detection and Alerting	52
4.4	Scheduled Task/Job: Scheduled Task (T1053.005)	54
4.4.1	Additional Endpoint Configuration	54

4.4.2	Detection and Alerting	55
4.5	Windows Management Instrumentation (WMI) Event Subscription (T1546.003)	58
4.5.1	Additional Endpoint Configuration	58
4.5.2	Detection and Alerting	59
4.6	BITS Jobs (T1197)	61
4.6.1	Additional Endpoint Configuration	61
4.6.2	Detection and Alerting	61
4.7	Windows Service (T1543.003)	62
4.7.1	Additional Endpoint Configuration	62
4.7.2	Detection and Alerting	62
4.7.3	Observations	64
5	Additional Testing and Observations	65
5.1	TrickBot	65
5.2	AgentTesla	66
5.3	RedLineStealer	68
5.4	Remcos	68
5.5	QBot	69
5.6	Detecting Possible False Positives	69
5.7	Final Observations	71
5.8	Future Work	72
	Conclusion	75
	Bibliography	77
	A Contents of attachments	91

List of Figures

1.1	Main categories of the MITRE ATT&CK matrix	5
1.2	Categories listed in the Persistence tactic of MITRE ATT&CK	6
2.1	Top malware families from Q4 2023 analyzed by <i>Any.Run</i>	13
2.2	Top malware statistics in Q4 2024 by <i>Avast</i>	14
2.3	Top malware families of samples available on <i>MalwareBazaar</i> (as of March 2024)	14
2.4	Reverse-engineered source code of <i>AgentTesla</i> malware displaying attempt to exploit the <i>startup folder</i> to achieve persistence.	19
2.5	Reverse-engineered source code of <i>AgentTesla</i> malware displaying attempt to exploit the <i>registry run key</i> to achieve persistence.	19
2.6	Reverse-engineered source code of <i>njRAT</i> malware displaying attempt to exploit the <i>startup folder</i> to achieve persistence.	20
2.7	Reverse-engineered source code of <i>njRAT</i> malware displaying attempt to exploit the <i>registry run keys</i> to achieve persistence. Note that both <i>HKCU</i> and <i>HKLM</i> registry hives are targeted.	20
2.8	Reverse-engineered source code of <i>njRAT</i> malware displaying attempt to exploit the <i>registry run key</i> to achieve persistence. Variable with the path to the particular key is visible here.	20
2.9	Reverse-engineered command used in a <i>RedLine</i> malware sample displaying an attempt to abuse the <i>schtask.exe</i> utility to achieve persistence.	24
2.10	A sample MOF file content defined to achieve persistence as shown by <i>PentestLabs</i>	27
2.11	Sample of the contents of a <i>MOF</i> file used by a malware sample belonging to the Mocking Bird group to gain persistence through WMI	28
2.12	Series of <i>bitsadmin</i> commands used by <i>UBoatRAT</i> to achieve persistence.	31

2.13	Utilization of <i>sc</i> to create a persistent Windows service by a Blue Mockingbird coin miner sample as detected by RedCanary.	33
2.14	Utilization of <i>sc</i> to create a persistent Windows service by a Ragnar Locker ransomware sample as detected by Sophos.	34
3.1	High level scheme of laboratory in Azure Cloud.	36
3.2	Departmental deployment scheme of Splunk.	38
3.3	Scheme of the laboratory environment in Azure with concrete technologies and subnets highlighted.	39
4.1	Design of the alert workflow between Splunk, GRR and Windows endpoint machine.	45
4.2	Design of the user interface of custom alert action to tie the alert with GRR action. One parameter is passed to define which action should be taken through GRR.	46
4.3	Enabling the <i>audit registry</i> policy using the Local Security Policy editor (<i>secpol.msc</i>).	49
4.4	Selecting the activities to monitor in connection with global registry SACL policy.	50
4.5	SACL policy to enable monitoring of the two default startup folders.	51
4.6	Policy setting to enable logging of whole commands in command line for event ID <i>4688</i>	51
4.7	Policy setting to enable PowerShell script block logging in order to invoke event ID <i>4104</i>	52
4.8	Enabling security audit logs of <i>Other Object Access Events</i> in order to invoke event ID <i>4698</i>	55
4.9	A command located inside an example scheduled task configuration XML logged by the event ID <i>4698</i>	56
4.10	Example of the data collected by GRR from the command line test case for detection of potential scheduled tasks abuse.	58
4.11	Enabling the Windows event ID <i>4697</i> via the group policy editor.	63
5.1	Alerts triggered after running a sample of the TrickBot malware.	65
5.2	Alerts triggered after running a sample of the AgentTesla malware.	66
5.3	Sample of one of the flows collecting artifacts from the test run of the AgentTesla malware.	67
5.4	Alerts triggered after running a sample of the Remcos malware.	68

List of Tables

5.1	Service File Names observed through the installation of Google Chrome	70
-----	--	----

Introduction

In the world where digitalization is rapidly growing, computer security is a topic concerning every one of us, whether we are using our devices for work related or personal purposes. The amount of cyber-attacks is continuously growing, there are many types, techniques and strategies for performing such attacks. The main objective for an attacker targeting a general user is being able to insert and run a malicious piece of software on the victim's machine. Even when the attacker is successful using any method, the same problem arises. What is next?

There are many things that the adversary has to keep in mind so that their attack is successful even after completing the initial step. One of those is making sure that their malicious code is run again and again undetected, even after the victim device is shut down and turned on again. This allows the attacker to maintain long-term access to the compromised system, enabling continued reconnaissance, data exfiltration, and attempts to further infiltrate into the network even after the initial access is lost. It grants the attacker the time and opportunity to achieve their objectives in the long run, whether they be theft of sensitive information, disruption of services, or sabotage of critical systems. There are many methods attackers use to achieve this very important attack stage. These methods are the main focus of this thesis, where the most common ones are discussed and analyzed.

Fortunately, it is possible to take advantage of this knowledge and use it to detect the malicious activity. This is especially true in an organizational environment where security monitoring tools are (or make sense to be) implemented. If suitable tools and strategy are used, malicious software attempting to gain persistence or an already persistent malware can be detected in real-time. Moreover, the detection can be enriched with additional relevant information that can be useful for further investigation.

The goals of the thesis are to investigate various methods of malware persistence, discover their principles and working mechanisms and explore the frequency of their occurrence based on research of different malware families

and various sources. The focus is on the machines running on Windows OS as the majority of real-world end users use this platform on their private and work devices. The objective of the practical part is to implement a monitoring solution with real-time detection of persistence based on the previous research. To achieve this, various tools are employed in a cloud laboratory environment and a set of detection rules is created along with necessary configurations for the laboratory devices. Lastly, the aim of the laboratory environment is to include a feature of automatic artifact acquisition to enrich the investigation data right away without the necessity to manipulate directly with the victim device.

Theoretical Background

This chapter aims to provide the theoretical foundation necessary to build a successful persistence detection environment. It helps organize the information so that it can be built on later and used practically. It also covers basic concepts and ideas that are necessary to be familiar with for later development, as well as technical details that are directly used within both design and implementation of the practical solution.

1.1 Malware

The main focus of this work is on a specific technique used in malware development. However, it is necessary to define which pieces of software are classified as malware.

Malware is short for malicious software. Malware is developed by cyber-criminals with the intention of causing harm to their victims, often involving stealing data or damaging target systems. Malware is divided into many categories according to its objectives and also into different families. [1]

1.1.1 Malware Families

Apart from the basic categories, malware can be divided into different families. A malware family is defined by the similarities of different pieces of software. It is possible to classify malware based on attack techniques, similar characteristics, or authors. These characteristics can later be useful for malware detection and investigation of malicious activity. As mentioned, each malware family often uses different mechanisms which can be analyzed and transformed into signatures that are later used for automatic detections.[2]

Later in this work, it can be observed from specific examples regarding malware persistence techniques.

1.1.2 Objectives of Malware and Importance of Persistence

As defined above, malware is a *malicious* piece of software. Malware developers create it with objectives that cause harm to the victim for their own benefit. Concrete motivation can vary as many malware categories exist, however, it is usually for financial gain. The malicious software can install advertisements for the user to click on, exfiltrate sensitive information (credit card details, passwords, classified information from a government agency, etc.), use the endpoint as a botnet slave to perform DDoS attacks, or even encrypt the data and demand payment for decryption (ransomware).

There are many different categories, and describing them is out of the scope of this thesis. However, it is all malware and it is not easy to infect a device with malware, regardless of the category. Therefore, it is not surprising that all malware creators need to think about what happens *after* the successful infection and not only about the goal itself. It is important to ensure that it will not be necessary to infect the target anew because, again, that is not trivial. It is important that malicious software stays on the target undetected and fulfills the purpose again and again, no matter how the system configuration changes or if the target device is shut down and started again.

To achieve this, many different techniques were used by many different malware families. The term used to describe this type of malware behavior is *persistence*. In the following chapters, more detailed information and examples will be introduced.

1.2 Persistence

Let us define the term formally. MITRE ATT&CK framework uses following definition: "Persistence consists of techniques that adversaries use to keep access to systems across restarts, changed credentials, and other interruptions that could cut off their access. Techniques used for persistence include any access, action, or configuration changes that let them maintain their foothold on systems, such as replacing or hijacking legitimate code or adding startup code" [4]

In other words, *persistence* in cybersecurity means the ability of an adversary to maintain long-term access to the victim system. When successful, the threat actor can keep access despite disruptive actions that terminate running code or change the state of the system (most notably reboot) without being detected, thus becoming *persistent* in the system. In particular, to achieve persistence, the system configuration needs to be manipulated in the attacker's favor. [3, 5]

1.2.1 Persistence Techniques Classification

The further concern is to understand which techniques malware developers use to achieve persistence. The scope of this thesis has been narrowed to focus only on the Windows OS. The goal is to identify and differentiate the techniques in a way that each one can be examined separately and a set of its characteristics can be obtained. Based on these observations, the aim is to put together a list of indicators which show that the technique is used on a target machine. This list can later be used as a good resource when implementing a detection solution. There are various sources that tackle this problem.

1.2.1.1 MITRE ATT&CK Framework

MITRE ATT&CK is a knowledge base of different techniques, tactics and procedures of cyberattacks. Its content is based on observations of the cybersecurity community throughout the years and is constantly being updated. It contains many categories of malicious behavior (tactics) that can be connected to many different objectives, such as information gathering, data exfiltration, and other related to malware activity. Many concrete techniques are described in detail, and some examples are included in the knowledge base. It is worth mentioning that some techniques can belong to multiple tactics. [6, 7]



Figure 1.1: Main categories of the MITRE ATT&CK matrix

[12]

The matrix is often used by real-world organizations to categorize detected malicious behavior to enrich their detection techniques.

1. THEORETICAL BACKGROUND

As mentioned, there is much information related to malware behavior, including the category of persistence. It contains 20 main categories of persistence methods, each divided into concrete techniques. Each technique is described to a different level of detail, examples of different malware families using the technique can be found there, as well as some mitigation and detection techniques.

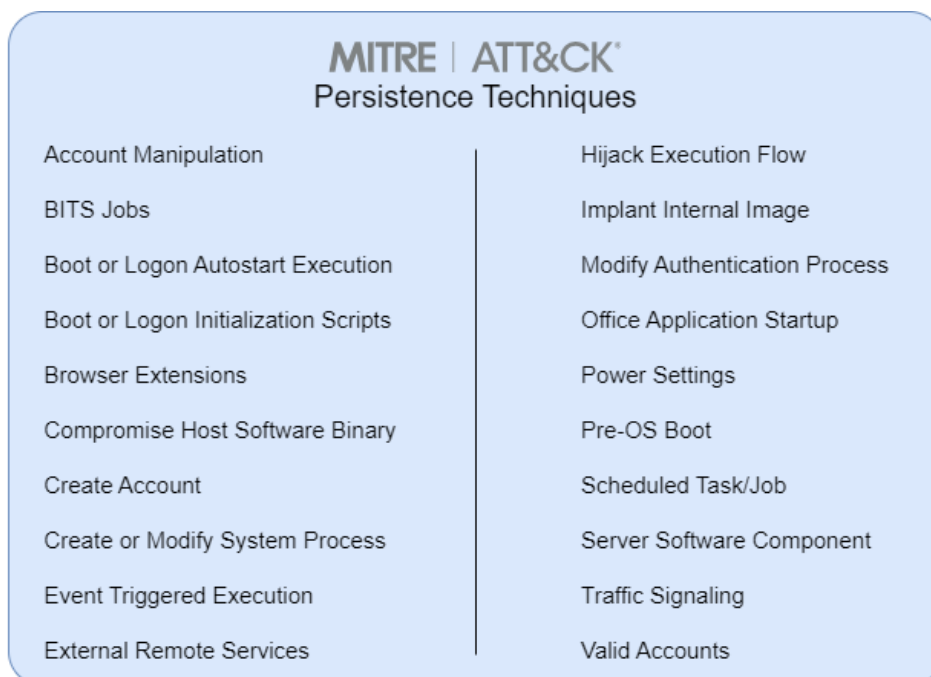


Figure 1.2: Categories listed in the Persistence tactic of MITRE ATT&CK

1.2.1.2 Other Classification Alternatives

There are not many publications or sources that deal with the problem of malware persistence classification. Some papers are available, but only one of them provides findings that can be useful for the goal of this work.

Villalón-Huerta, Marco-Gisbert, and Ripoll-Ripoll create a taxonomy of malware persistence techniques regardless of its targeted platform. Their taxonomy relies strongly on the use of the MITRE ATT&CK matrix. However, they do not dive deep into individual techniques, as they rather build an abstract taxonomy on top of them. [8]

In a thesis written by Webb, basic classification is also introduced, but again without a complex breakdown of individual techniques. [11]

A recent work by van Nielen, however, provides an overview of 70 malware persistence techniques (also focusing solely on Windows OS). He conducts research of various resources available on the Internet: blog posts, technical articles, and academic publications. Similarly to other authors who deal with the topic of malware persistence, he makes strong use of the MITRE ATT&CK knowledge base. Many of the listed techniques align with the framework, however, the list is enriched with techniques described elsewhere. The list provides a very brief description of each individual technique.

Moreover, he provides an enriched taxonomy of the persistence techniques, adding requirements which are necessary for each technique to work (*filesystem write, registry write, elevated privilege, interprocess communication, additional software*), the effects that the achieved persistence has on the system and the place where the techniques reside in the system (*databases and file system*). The former, especially, can be useful to understand individual techniques from a higher-level perspective when designing detection mechanisms.

The paper concludes with a part that studies the frequency of occurrence of different persistence techniques. The dataset includes 5000 malware samples that are tested through a custom detection model with the goal of detecting different persistence mechanisms. Although the author admits that there can be some statistical deviations and undetected samples, it gives a good overview of the most commonly used persistence techniques. Combined with other sources, this can be a very valuable resource for selecting relevant persistence techniques for detection.

He, however, conducts the detection of persistence for his research in a laboratory environment while utilizing binary instrumentation. It would be very challenging to deploy this method in a real-world company scenario where real-time detection is desired, the main problem being modification of every launched application. This method would have a significant impact on system resources. [10]

The publications described above try to bring something new regarding persistence classification or taxonomy. However, unsurprisingly, they all rely strongly on the MITRE ATT&CK framework, as they mostly introduce new categorizations on top of the framework or add new elements which extend the framework.

1.3 Forensic Artifacts

After detecting a potential threat on a system, it is important to examine what exactly happened, to determine whether it is indeed a result of malicious activity or only a false positive. In case it turns out that the system is compromised, it is crucial to discover how the compromise occurred as well as what is the impact. To achieve this, it is necessary to obtain relevant *forensic material*.

Focusing on endpoint devices, key *forensic artifacts* include various files, logs, and configurations stored on the endpoint device. These can incorporate for example system logs, event logs, registry entries, file system metadata, and more information about the system. [47]

In this work, the scope includes only Windows endpoints. For later utilization, it is worth diving into the most important artifacts that can be acquired from a Windows device. There are many different kinds of artifacts associated with persistence techniques, however, the two described in this section are the most complex and common ones as well as relevant for the majority of techniques not only in connection to persistence. It is therefore worth to give them special attention before utilizing them practically.

1.3.1 Windows Event Logs

This section relies mostly on the following sources: [48, 49, 50, 51, 52]

Windows event logs are very helpful when it comes to forensic analysis. It logs course of actions that occurred on the system with timestamps, which helps the analyst put information into context. Various types of system, user, or application activities are logged and these events are divided into the following categories: *information, warning, error, critical, and success/failure audit*.

There are five areas of events that are recorded by the Windows systems:

- **Application** - Events that are related to applications installed on the local system.
- **Security** - Contains events related to security events according to the auditing policy. That can include for example login attempts, elevated privileges or file deletions.
- **Setup** - This area contains events that occur during the installation or upgrade of the Windows operating system and logs its enterprise features.
- **System** - Logs that are generated by the operated system itself, for example logs about device drivers.
- **Forwarded** - Contains event logs forwarded from other computers in the same network in a specifically configured environment.

Aside from these default Windows categories and their corresponding files, there are also log files created for individual applications and components, which can be found at the same location. These can also be very useful for forensic analysis. The logs can be typically found in the following directory:

```
C:\Windows\System32\winevt\Logs
```

Log files typically end with *.evt* extension and have a defined structure. They can be viewed, for example, by a native Windows program called *Event Viewer* as well as by other open-source or proprietary software.

1.3.2 Windows Registry

This section relies on the following sources: [53, 54, 55, 56, 57, 58, 59, 60]

Windows registry is a crucial forensic artifact. It is a collection of databases which store different system and user settings, application configuration data, and other values important for the Windows operating system. As there are many kinds of information stored in the registry, it can provide the analyst with many helpful information and evidence including, for example, evidence of launched applications, various user actions, unauthorized system changes, and much more data that can even prove the presence of a malware.

This basic registry overview will later help give context to behavior that can be employed by some persistence techniques and serve as a reference for selecting relevant artifacts connected to the specific technique.

The registry is organized into so-called *registry hives*. As defined by Microsoft, "A *hive* is a logical group of keys, subkeys, and values in the registry that has a set of supporting files loaded into memory when the operating system is started or a user logs in." [55] Each "hive" has a structure resembling folders, where every key acts like a "folder". Each key can contain a subtree of subkeys, and each key can also contain a value. Typically, there are 5 *registry hives* on a Windows machine designated to store different types of information or configuration:

- **HKEY_CLASSES_ROOT** - This hive stores configuration of which application is used for opening particular type of files.
- **HKEY_CURRENT_USER** - User settings are included here, it is tied to a specific user. The file containing the information itself is located within the folder of a particular user.
- **HKEY_LOCAL_MACHINE** - This is a hive containing critical system information, including hardware and software or security settings.
- **HKEY_USERS** - Contains information about more than one users logged on.
- **HKEY_CURRENT_CONFIG** - The purpose of this hive is to store information for real-time measurement.

It is important to note that the above mentioned classification is completely applicable only while working on a live system. The situation is different when dealing with a disk copy or when identifying relevant isolated

1. THEORETICAL BACKGROUND

artifacts for analysis. Let us focus on hives that contain relevant information for analysis and their corresponding files:

- The **HKEY_LOCAL_MACHINE** (HKLM) hive is certainly very interesting, as it contains many system information. There are four "sub-hives" under HKLM: *SAM*, *SECURITY*, *SOFTWARE* and *SYSTEM*. They are located in the following directory:

C:\Windows\System32\Config

and have corresponding filenames with no file extension.

- Information about the user is also a valuable resource. The file matching the **HKEY_CURRENT_USER** (HKCU) hive is located in the user directory. Each user has their own file in their corresponding directory:

C:\Users\<<username>\NTUSER.DAT

- Another registry file which also contains interesting information about the user is represented as **HKEY_CURRENT_USER\Software\CLASSES** on a live system and is located as follows:

C:\Users\<<username>\AppData\Local\Microsoft\Windows\USRCLASS.DAT

It is also relevant to mention that there are different privileges needed to change values inside the system and the user registry hives. System hives such as *HKLM* require *SYSTEM* privileges, whereas user hives such as *HKCU* only need privileges of the corresponding user. [84]

In addition to the registry files themselves, it is worth mentioning the existence of registry log files. These files can typically be found in the same directory as the registry files with corresponding names with the extension *.LOG1* or *.LOG2* (Windows may use 2 different log files at the same time). These files can be useful, for example, for recovering deleted registry entries (however, as it is generally common for log files, the oldest data are overwritten by new data, the information can therefore be limited).

There is one more registry hive that can be interesting to examine and recover. It is called *Amcache* and stores information related to program executions. The file can be found at the following location:

C:\Windows\appcompat\Programs\Amcache.hve

Exploring Persistence Techniques

In this section, different persistence techniques are examined thoroughly. Before that, it is crucial to define which techniques make sense to be the subject of thorough analysis. It is necessary to dive into various sources to discover which techniques are used in the real world. The goal is to then understand how the techniques operate, so that detection methods can be designed for each of them. The analysis of the selected techniques should provide detailed information to serve as a guideline while implementing the practical detection solution.

2.1 Identifying Popular Persistence Techniques

There are multiple ways to perceive what *popular* means in relation to the selection of persistence techniques. However, we can identify two key standpoints:

- Frequency of occurrence of the technique in general - the goal is to identify commonly mentioned techniques used by malware samples in general regardless of malware family in various publications, eg. papers, reports, articles or blogposts.
- Persistence techniques used by currently trending malware families - if we are able to identify a list of recently active malware families, it is then possible to look at the techniques they are using and select the most common ones.

After examining both approaches, it should be possible to combine the results and finalize a list of frequently repeated techniques that will be regarded as relevant for including in the detection mechanisms in the practical solution and examined in greater detail.

Individual techniques are referred to as named in the MITRE ATT&CK matrix as it can be assumed that all identified techniques are included there. To clarify, the focus here is on particular sub-techniques as there can be significant technical differences between them. The focus is on techniques relevant to the Windows OS, reflecting the scope of this work.

2.1.1 Frequently Referred Persistence Techniques

There are many papers and articles that describe particular techniques. However, there are not many studies and statistics regarding this topic that would provide us with direct insight. Upon examination of different publications, there are several techniques that repeatedly stand out.

Elastic Security Labs, for example, list three popular persistence techniques: *Windows Management Instrumentation (WMI) Event Subscription (T1546.003)*, *BITS Jobs (T1197)* and *Scheduled Task/Job (T1053)*. [15]

The Red Report is a stat report published by Picos Security every year and it is based on analysis of tens of thousands of malware samples. The publication identifies Top 10 general MITRE ATT&CK techniques. In the last two years, only two persistence techniques appeared there, being *Scheduled Task/Job (T1053)* and *Boot or Logon Autostart Execution (T1547)*. Note that both of them contain multiple sub-techniques. [14] [13]

A statistic published by AnyRun ¹ regarding the 4th quarter of 2023 also lists two persistence techniques among the most used by submitted malware samples - *Scheduled Task/Job: Scheduled Task (T1053.005)* and *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)* [20]

While examining other articles and publications, other additional or concrete sub-techniques stood out: *AppInit DLLs (T1546.010)*, *DLL Search Order Hijacking (T1574.001)*, *DLL Side-Loading (T1574.002)* and *Windows Service (T1543.003)*. [16, 17, 18, 19]

Moreover, the previously mentioned (1.2.1.2) recent publication by van Nielen includes a graph that summarizes the frequency of persistence methods based on 5000 samples. It shows that amongst the most frequent techniques are *Dll Hijack* (including both T1574.001 and T1574.002), *Registry Run Keys/Startup Folder (T1547.001)*, *Windows Management Instrumentation (WMI) Event Subscription (T1546.003)*, and also *Windows Service (T1543.003)*. [10]

2.1.2 Techniques Within Popular Malware Families

We can rely on some interesting statistics while tackling the perspective of exploring common malware families. One can come across multiple available resources based on objectively measurable data. These statistics can be based

¹AnyRun is a public sandbox where users can upload suspicious files and conduct dynamic malware analysis automatically.

on both samples collected by the security community or research conducted by private companies.

Previously mentioned *Any.Run* is a widely used sandbox service that publishes various statistics on a regular basis. It is useful to explore the data that are available from these types of sandboxes. The data are interesting mainly because samples analyzed by the sandbox are submitted by a wide range of users, the majority being malware analysts and various cybersecurity professionals. Therefore, it can be argued that these statistics include *real* samples that are found *in-the-wild*².

Naturally, the popularity of different malware families is continuously varying, which is reflected in the reports, as they are typically published for each quarter of the year. Regarding the time period of creation of this work, the *fourth quarter of 2023* is considered the most recent and relevant. The most popular malware families as detected by *Any.Run* for Q4 2023 are shown in figure 2.1. [20]



Figure 2.1: Top malware families from Q4 2023 analyzed by *Any.Run* [20]

Established companies such as *Avast* conduct their own research and publish periodic reports. The focus is again on the report that describes the fourth quarter of 2023. In the report, they divide malware into 3 main categories: *coin miner*, *information stealer*, and *ransomware*. In table 2.2, the most frequently detected families by Avast are shown.

Lastly, it is worth looking at another statistic consisting of samples col-

²In-the-wild threats are threats spreading among real world computers - as opposed to test systems[21]

2. EXPLORING PERSISTENCE TECHNIQUES

Coinminer	Information Stealer	Ransomware
XMRig (63%)	AgentTesla (26%)	STOP (17%)
Web miners (19%)	FormBook (10%)	WannaCry (16%)
CoinBitMiner (2%)	Fareit (6%)	Enigma (9%)
SilentCryptoMiner (2%)	RedLine (4%)	TargetCompany (4%)
FakeKMSminer (1%)	Lokibot (3%)	Cryptonite (2%)
NeoScript (1%)	Lumma (3%)	LockBit (1%)
CoinHelper (1%)	Stealc (2%)	
	OutSteel (2%)	
	ViperSoftX (2%)	
	Raccoon (2%)	

Figure 2.2: Top malware statistics in Q4 2024 by *Avast*

[22]

lected by the cybersecurity community. *MalwareBazaar*³ provides statistics of all malware samples shared on the platform. The chart showing the top malware families as can be seen in figure 2.3 is this time not limited to the fourth quarter of 2023 as it is being constantly updated. The graph shows the data as of March 2024.

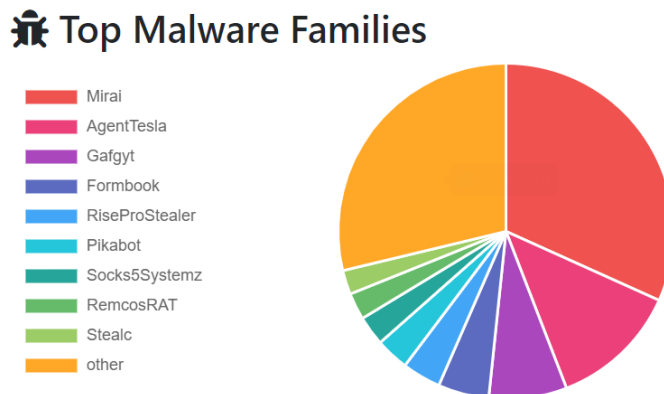


Figure 2.3: Top malware families of samples available on *MalwareBazaar* (as of March 2024)

[23]

Looking at the sources mentioned above, it can be seen that, surprisingly, the results differ. In spite of that, 5 (Windows only) families can be identified as predominant:

³”MalwareBazaar is a project from abuse.ch with the goal of sharing malware samples with the infosec community, AV vendors and threat intelligence providers.” [23]

- *AgentTesla* is very common according to all three sources.
- *RedLine* appears in both *Any.Run* and *Avast* statistics.
- *Formbook* can also be found in all three graphs.
- *Remcos* is third among the *Any.Run* detections and appears also in the *Malwarebazaar* graph.
- *Stealc* was included in the top ten by both *Avast* and *Malwarebazaar*.

2.1.2.1 Examining Selected Families

After selecting representatives of recently trending malware families, let us explore the persistence techniques they use.

- **AgentTesla** - according to multiple sources, its most common persistence technique is *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)*. *Scheduled Task/Job: Scheduled Task (T1053.005)* is also among the used techniques. [25, 26, 27, 9]
- **RedLine** - Similarly to *AgentTesla*, *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)* and *Scheduled Task/Job: Scheduled Task (T1053.005)* are the techniques employed for persistence according to various write-ups. [28, 29, 30, 31, 32]
- **Formbook** - According to multiple analyses, *Formbook* employs the same techniques as the two families above. [36, 35, 34, 33]
- **Remcos** - *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)* is the main technique used by this family. [37, 38, 39, 40]
- **Stealc** - There are not many resources that would describe persistence techniques of *Stealc*. According to a *Tria.ge* sandbox result, the persistence technique employed is *Scheduled Task/Job: Scheduled Task (T1053.005)*. [41]

2.1.3 Observations

As seen in the section above, both current trends and historical experience (as seen in various articles about persistence techniques mentioned in 2.1.1) indicate that *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)* and *Scheduled Task/Job: Scheduled Task (T1053.005)* are highly popular techniques used for persistence, it is therefore certainly necessary to include them in the detection solution.

Moreover, it is very common for *fileless malware*⁴ to establish persistence using registry entries. Detection of this technique is therefore beneficial while hunting also for this type of malware. [46, 45]

However, other techniques can't be overlooked because they certainly are historically used in major malware families. *BITS Jobs (T1197)* is, for example, used by widespread malware family *QakBot* or by *APT47*. [15]

Dll Hijack (T1574.001 and T1574.002) stands atop of Van Nielen's measurement and is also employed by a large number of malware families. [10, 43]

Windows Management Instrumentation (WMI) Event Subscription (T1546.003) also appeared in a higher number of samples and is employed for example by the infamous *Lazarus Group* or by *APT41*. It is also a technique frequently employed by previously mentioned *fileless malware*. [10, 42, 44]

Historically, *Windows Service (T1543.003)* is also a widely used technique.

2.2 Detailed Overview of Persistence Techniques

In this section, individual persistence techniques are examined in detail in order to understand their technical principles and identify detection opportunities. Some examples of the techniques used in real-world malware samples are also shown. The information are later used in the implementation of the practical detection mechanisms.

The goal is also to identify artifacts significant to each persistence technique. The focus is on additional artifacts on the endpoint that can be collected remotely and that are useful to the analyst examining the alert, in order to enrich the information available from logs. That can help the analyst in an effective analysis of the alert. Of course, there are many artifacts that would help the analyst. However, the goal is not to collect all the forensic data available just to investigate one alert, the intention is to collect most relevant and accessible artifacts that can be acquired remotely and quickly for basic yet enhanced alert investigation. If the investigation indicates that the alert is a true positive, the whole forensic copy can be acquired for an in-depth analysis during the incident response procedure.

2.2.1 Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)

As concluded in the previous section, this technique is highly popular among malware families. Technically, there are two techniques combined, however, the principle is pretty similar as utilizing both registry keys as well as the

⁴"Fileless malware is a type of malicious activity that uses native, legitimate tools built into a system to execute a cyber attack. Unlike traditional malware, fileless malware does not require an attacker to install any code on a target's system, making it hard to detect." [45]

startup folder results in execution of a program after user logon. Moreover, the startup folder settings are tied to registry values.

2.2.1.1 Principle

When it comes to how attackers take advantage of the *startup folder*, it is not overly complicated. There are two locations where, if a file or a shortcut is placed there, the corresponding application launches after user logon: [65, 63, 61]

- Directory with programs set to start for the specific user:
`C:\Users\[Username]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\StartUp`
- Applications placed there will start regardless of which user logs in:
`C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp`

Therefore, it is only necessary for a malicious program to place the intended file in one of these directories, and it will be executed after each logon, resulting in gaining persistence.

However, the particular location can be modified and that is when the registry comes in. There are four registry keys that carry the value of the location of the *Startup* folders mentioned above (also called shell folders). The default locations as mentioned can be changed by editing the values of these registry keys: [63, 61, 65]

- `HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders`
- `HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders`
- `HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders`
- `HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders`

It can be confusing why there are 4 entries that define the location of the startup folder. As explained in Microsoft documentation: *"The entries that appear in User Shell Folders take precedence over those in Shell Folders. The entries that appear in HKEY_CURRENT_USER take precedence over those in HKEY_LOCAL_MACHINE."* [63]

There, however, exist also *registry keys* which include settings for direct autostart of an application. There are 4 fundamental registry keys, which are capable of setting a program to start after each logon: [64, 65, 62]

2. EXPLORING PERSISTENCE TECHNIQUES

- *HKCU\Software\Microsoft\Windows\CurrentVersion\Run*
- *HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce*
- *HKLM\Software\Microsoft\Windows\CurrentVersion\Run*
- *HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce*

By adding an entry to one of these keys, persistence is achieved, as the desired application starts after user logon. Entries in the *HKCU* hive are associated with logon of the corresponding user while the *HKLM* keys affect all users of the system. It is important to note that whilst the *Run* key entry stays on the machine the whole time, the *RunOnce* key entry is deleted after launching the desired program. [62, 64]

A similar effect can be achieved by adding an entry to the keys connected with the policy settings: [65]

- *HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run*
- *HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run*

The *load* value of the following entry can also be exploited for persistence: *HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows* [65]

The last set of registry keys observed to be exploitable for persistence is connected with *Windows services*. The following 4 keys can control the autostart of an already existent *Windows service*: [65]

- *HKCU\Software\Microsoft\Windows\CurrentVersion\RunServices*
- *HKCU\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce*
- *HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices*
- *HKLM\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce*

2.2.1.2 Examples

The examples below show the real usage of both, exploiting the registry keys as well as the startup folder. As seen, some malware samples are capable of both.

AgentTesla

As mentioned earlier, *AgentTesla* malware family uses this technique of persistence. As concluded from an analysis by Splunk, AgentTesla is capable of attempting to use both *startup folder* and *registry run keys* for establishing persistence. Figure 2.4 and figure 2.5 show concrete code snippets. [26]

```

private static void B()
{
    if (global::A.C.A.d && Operators.CompareString(global::A.C.A.c, global::A.C.A.B, false) != 0)
    {
        if (!Directory.Exists(Environment.GetEnvironmentVariable("%startupfolder%") + "\\%insfolder%\\"))
        {
            Directory.CreateDirectory(Environment.GetEnvironmentVariable("%startupfolder%") + "\\%insfolder%\\");
        }
        try
        {
            if (File.Exists(global::A.C.A.B))
            {
                try
                {
                    string fullPath = Path.GetFullPath(global::A.C.A.B);
                    foreach (Process process in Process.GetProcesses())
                    {
                        string fullPath2 = Path.GetFullPath(process.MainModule.FileName);
                        if (Operators.CompareString(fullPath2, fullPath, false) == 0)
                        {
                            process.Kill();
                        }
                    }
                }
                catch (Exception ex)
                {
                }
            }
            if (File.Exists(global::A.C.A.c))
            {
                if (File.Exists(global::A.C.A.B))
                {
                    try
                    {
                        File.Delete(global::A.C.A.B);
                    }
                    catch (Exception ex2)
                    {
                    }
                }
                File.Copy(global::A.C.A.c, global::A.C.A.B, true);
                if (global::A.C.A.E)
            }
        }
    }
}

```

Figure 2.4: Reverse-engineered source code of *AgentTesla* malware displaying attempt to exploit the *startup folder* to achieve persistence.

[26]

```

try
{
    using (RegistryKey registryKey = RegistryKey.OpenBaseKey(RegistryHive.RegistryView.Default, OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true))
    {
        registryKey.SetValue(path.GetFileNameWithoutExtension(A_0), "\"" + A_0 + "\"");
    }
    flag = true;
}
catch
{
}
try
{
    using (RegistryKey registryKey2 = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Default).OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon", true))
    {
        string text = (string)registryKey2.GetValue("Shell");
        registryKey2.SetValue("Shell", text + ", \"" + A_0 + "\"");
    }
    flag = true;
}

```

Figure 2.5: Reverse-engineered source code of *AgentTesla* malware displaying attempt to exploit the *registry run key* to achieve persistence.

[26]

2. EXPLORING PERSISTENCE TECHNIQUES

njRAT

Malware called *njRAT* uses similar approach to *AgentTesla* with establishing persistence using registry keys and the startup folder. It is as able to utilize both *startup folder* and *registry run key*. [66, 67]

```
539         if (OK.IsF)
540         {
541             try
542             {
543                 File.Copy(OK.LO.FullName, Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\\" + OK.RG + ".exe", true);
544                 OK.FS = new FileStream(Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\\" + OK.RG + ".exe", FileMode.Open);
545             }
546             catch (Exception ex6)
547             {
548             }
549         }
```

Figure 2.6: Reverse-engineered source code of *njRAT* malware displaying attempt to exploit the *startup folder* to achieve persistence.

[66]

```
flag = OK.Isu;
if (flag)
{
    try
    {
        OK.F.Registry.CurrentUser.OpenSubKey(OK.sf, true).SetValue(OK.RG, "\"" + OK.LO.FullName + "\" ..");
    }
    catch (Exception ex3)
    {
    }
    try
    {
        OK.F.Registry.LocalMachine.OpenSubKey(OK.sf, true).SetValue(OK.RG, "\"" + OK.LO.FullName + "\" ..");
    }
    catch (Exception ex4)
    {
    }
}
```

Figure 2.7: Reverse-engineered source code of *njRAT* malware displaying attempt to exploit the *registry run keys* to achieve persistence. Note that both *HKCU* and *HKLM* registry hives are targeted.

[67]

```
525         {
526             OK.F.Registry.CurrentUser.OpenSubKey(OK.sf, true).SetValue(OK.RG, "\"" + OK.LO.FullName + "\" ..");
527         }
1706         // Token: 0x04000015 RID: 21
1707         public static string sf = "Software\\Microsoft\\Windows\\CurrentVersion\\Run";
```

Figure 2.8: Reverse-engineered source code of *njRAT* malware displaying attempt to exploit the *registry run key* to achieve persistence. Variable with the path to the particular key is visible here.

[66]

Ryuk

A known ransomware named *Ryuk* also achieves persistence through the *registry run key*. However, unlike the two families mentioned above that are written

in .NET, it uses following command to establish the registry key entry: [68]

```
C:\Windows\System32\cmd.exe /C REG ADD
"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows
\CurrentVersion\Run" /v "svchos" /t REG_SZ /d
"C:\users\Public\BPWPc.exe" /f
```

2.2.1.3 Detection

When it comes to detecting the *registry run key* exploitation, the main goal is to monitor changes of the previously mentioned registry keys. One of the methods how to achieve that utilizes the feature of native *Windows event logs* as described in section 1.3.1.

Information about a change of a registry value is recorded in an event with ID *4657*. The event is part of the **Security** audit logs which are disabled by default, is it therefore necessary to enable it for example by setting a group policy. Aside from general information about the activity, such as the *username* or the *name of the program* which invoked the event, it contains the **Object name** entry, which records the exact name of the modified registry key and the **Object value name**, which carries information about the concrete value set to the modified key. Both of these entries are valuable indicators for determining whether there was an attempt to gain persistence. If the **Object name** contains one of the registry keys mentioned in 2.2.1.1, it is an indicator that the source machine might be compromised. Of course, false positives can occur as some legitimate applications may use these registry values for its proper functionality. [69]

There are various other tools that can monitor and log registry modifications. When designing a practical solution, they must be taken into account alongside the mentioned native log solution, as they might be more suitable depending on the particular environment. However, the goal stays the same: **detect modification of the identified registry keys**.

Other effective method of detecting registry modifications connected to malicious activity could be monitoring the command line and Powershell. This method would be effective for the *Ryuk* example mentioned in the previous section. A native *Windows event log* with ID *4688* can be utilized for this purpose when enabled with appropriate settings. [70, 71, 83]

Detecting modification of files in the startup folder can be achieved in a similar way through the *Windows event logs*. For example, it is possible to utilize event ID *4656*. As described in Microsoft documentation: *"This event indicates that specific access was requested for an object. The object could be a file system, kernel, or registry object, or a file system object on removable storage or a device."* Therefore, if the **Object name** entry of the logs includes one of the paths to the startup folders, an alert can be generated. Similarly to the registry keys, false positives can occur. However, when enabled, this event

is generated very frequently and it must be taken into consideration whether it is worth it to send and store so many log entries. [72]

2.2.1.4 Artifacts

The easiest and crucial artifact to obtain are logs. If there is a SIEM or log management solution in place and the logs are stored somewhere, this artefact is already obtained. Otherwise it is necessary to acquire them separately. For this type of events, the analyst can learn a lot of information from the previously mentioned *Windows event logs*. As described in 1.3.1, the logs are stored as *.evtx* files located in a specific directory. This applies to all the techniques, it is only necessary to select particular logs.

When it comes to artifacts related to manipulation with the contents of the startup folders, it would be desirable to obtain the concrete file (or files) that was added. Let us not forget that a shortcut file can also be used, in that case, the ideal scenario would be to obtain the file the shortcut is pointing to.

When a registry modification is detected, it is mainly necessary to identify which of the registry keys was modified and which value was added. That information can be observed from the log with ID *4657* as mentioned previously. From there, it could be possible to extract the file set to be launched.

Most importantly, the key artifact applicable for both of these cases (and arguably for other techniques as well) would be to obtain the particular file (eg. exe, malicious document or a different binary) which initially started performing the potential malicious activity. A windows event ID *4688* carries information about newly started processes, including *process ID*, *name* and *parent process name* which also includes the path to the launched file. It would be therefore possible to correlate the event with *4657* through process ID, which is present in both. [83]

2.2.2 Scheduled Task/Job: Scheduled Task (T1053.005)

Another very widespread technique utilizes scheduled task, a native Windows functionality. According to multiple reports, it is one of the most popular persistence techniques in recent years. [14, 13, 20]

2.2.2.1 Principle

Scheduled tasks are a handy utility for legitimate use and many harmless programs make use of it. The purpose of the utility is to execute routine tasks on a device when certain conditions (triggers) are met. A trigger can be for example a specific time, or a specific event on the machine. [73]

That is also very interesting for an attacker. By using the scheduled tasks feature, adversaries are able to execute malicious tasks regularly and automatically. With scheduled tasks surviving system reboot, it suits very well for the purpose of persistence.

There are multiple ways to set up a scheduled task. The most common method used by malicious programs native command line utility *schtasks.exe*. There are multiple arguments that can be utilized to configure the parameters of the scheduled task. A typical simple command for creating a scheduled task for notepad can for example look like this:

```
schtasks /CREATE /SC DAILY /ST 15:00 /TN "Persistent
Notepad" /TR C:\Windows\System32\notepad.exe"
```

The above command creates the scheduled tasks, sets daily recurrence, start time, name of the task and path to the target binary to launch. There are also other parameters that can be abused by attackers instead of only */Create* (*/Change*, */Run*, */Delete*, and */Query*). [75, 76, 77, 78]

It should be noted that it is possible to achieve the same effect with cmdlet *Set-ScheduledTask* in Powershell. [79]

Aside from the command line options, there have been cases of utilizing .NET wrapper for Windows task scheduler or netapi32 library for creating scheduled tasks. [78]

After the scheduled task is created, an XML file with its configuration is created in *C:\Windows\System32\Tasks*. [75, 74]

2.2.2.2 Examples

There are two examples selected to show the exploitation using scheduled tasks. They illustrate that the command line is a popular method of setting up persistence through scheduled tasks.

AgentTesla

The *AgentTesla* family is capable of utilizing both registry and scheduled tasks for persistence. According to an analysis published by Osama Ellahi, it performs the following command:

```
schtasks.exe /Create /TN "Updates\kzsAJcIeUIa" /XML
"C:\Users\%username%\AppData\Local\Temp\tmp95EB.tmp"
```

The task settings are saved in the tmp95EB.tmp file, which is an XML file loaded by the command. Note that the file name and the task name can be generated differently between individual samples. [80]

RedLine Stealer

RedLine also uses the *schtasks.exe* tool to establish persistence, as concluded from analysis by Gridinsoft.

In this sample (figure 2.9), the RedLine malware does not use an XML file to provide configuration, it is provided immediately as arguments. The name of the scheduled task is *Puoi* with the execution file saved in the *AppData*

```
schtasks.exe /create /tn "Puoi" /tr "C:\\Users\\user\\AppData\\Local\\Temp\\zqNDtAgMrV\\PJKIgrDmM.exe.com C:\\Users\\user\\AppData\\Local\\Temp\\zqNDtAgMrV\\z" /sc minute /mo 3 /F
```

Figure 2.9: Reverse-engineered command used in a *RedLine* malware sample displaying an attempt to abuse the *schtasks.exe* utility to achieve persistence.

[28]

location of the targeted user with the following path being used as the program argument. By */sc* and */mo* flags, the task is set to launch every 3 minutes. The */F* flag forces the task creation with surpassing warnings. [28, 77]

2.2.2.3 Detection

The most apparent detection method when looking at the principle is to **monitor the command line**. The general idea would be to monitor the executed command and if a *schtasks.exe* (or the *Set-ScheduledTask* cmdlet) is the command invoked, it can be an indicator of malicious activity. Naturally, false positives can occur, as there are legitimate applications (even in Windows by default) that can create new tasks. In a real environment, they can be later whitelisted.

Another effective way to detect scheduled tasks abuse is to monitor activity associated with scheduled task processes. This can be again achieved with native Windows event logs. One useful log source are the *Microsoft-Windows-Task-Scheduler/Operational* logs. Every time a scheduled task is created, events with IDs *106* and *140* are generated. It is also possible to utilize the *Security* logs, which monitor the creation of a new task with event ID *4698* when enabled. [76, 75]

Moreover, there is also the possibility of monitoring changes in the *C:\Windows\System32\Tasks* directory.

2.2.2.4 Artifacts

Similarly to the previous technique that abuses the registry, *logs* are an artifact carrying a lot of useful information for investigation. As described in previous section about detection, the main logs of interest in scheduled tasks activity are the *Windows-Task-Scheduler/Operational* logs and the *Security* logs (*106*, *140* and *4698*).

Specifically for this technique, it would be beneficial to recover files (ideally newly added files) from the the *C:\Windows\System32\Tasks* directory.

There are also interesting artifacts present in the registry. When a new task is created, new entries are added to the following registry keys: [81]

- *HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks*

- *HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree*

A very important artifact to acquire would be the command (and corresponding file) scheduled to launch. That information can be obtained both from the *Tasks* key entry and the XML configuration file present in the *Tasks* directory. It can come in handy that the Windows event ID *4698* contains the whole content of the XML configuration information. [81, 82].

Again, it would be ideal to obtain the originating process of this behavior. The ID of the process (and since the Windows 10 version 1903 also the parent process ID) that created the task can be obtained from the security event *4698*. Then it is possible to perform correlation with the event ID *4688*.

2.2.3 Windows Management Instrumentation (WMI) Event Subscription (T1546.003)

WMI is a native tool present in Windows systems designated to query information about the system, manage Windows computers, invoke methods, and perform certain actions on a particular system, often based on certain conditions. This makes WMI a useful tool for system administrators, however, it can also be utilized by adversaries to conduct attacks, hiding some useful artifacts. [98, 99]

WMI consists of the following basic features, which work together to form a database containing *namespaces*, where certain *classes* are included. There are many important native system classes described in the official Microsoft documentation, however, according to other sources, there are 3 main system classes necessary for basic functionality and also relevant to adversarial activity: [97, 100, 101]

- *__EventFilter* - Describes certain conditions which are to be met in order to invoke some action. In other words, it acts as a definition of a trigger.
- *EventConsumer* - Serves to perform a defined action, there are multiple pre-defined classes of *EventConsumer* present in Windows: (*ActiveScriptEventConsumer*, *CommandLineEventConsumer*, *LogFileEventConsumer*, *NTEventLogEventConsumer*, *ScriptingStandardConsumerSetting*, *SMTPEventConsumer*)
- *__FilterToConsumerBinding* - Ties together *EventFilters* and *EventConsumers*.

A file format used to define the WMI classes that are inserted into the database is called *MOF* (*Managed Object Format*).

2. EXPLORING PERSISTENCE TECHNIQUES

The entire WMI repository can be regarded as an artifact. It is located in `%SYSTEMROOT%\System32\wbem\Repository` and consists of following files: [102, 103]

- ***OBJECTS.DATA*** - Contains objects managed by WMI.
- ***INDEX.BTR*** - Serves as index of files imported into *OBJECTS.DATA*
- ***MAPPING[1-3].MAP*** - Exists to correlate data in *OBJECTS.DATA* and *INDEX.BTR*

WMI can be used not only for purposes of *Persistence*, it is also commonly leveraged for *Lateral Movement*. In comparison to the previous persistence techniques, it allows the attacker to have more control over the conditions when a given action is performed. WMI can react to various system conditions, not only a scheduled time or startup. [104]

2.2.3.1 Principle

In order to achieve persistence, the adversary has to introduce a new *WMI event subscription*. Practically, this means creating a certain *EventFilter* defining the conditions that must be met when the desired action is to be taken. That can be for example system startup, user login, or execution of a particular application. The filter is associated with one of the *EventConsumer* classes. For persistence, *CommandLineEventConsumer* is very suitable as it allows launching a command as a reaction to an event occurring. Moreover, *ActiveScriptEventConsumer* can be used to invoke a predefined script in an arbitrary scripting language on the system. Creating the consumer is key for the attacker, as that is what defines which actions are performed after the trigger. Lastly, it is necessary to set up the binding. [97, 104, 101]

In practice, there are multiple ways how an attacker can perform these three steps to achieve persistence. One possibility is to use a *MOF* file containing the definition of all three classes and then to compile it using a built-in windows tool *mofcomp.exe*, which automatically inserts the classes defined inside the *MOF* file into the WMI database. As shown in figure 2.10, the *MOF* file can define all three necessary classes to achieve persistence. It is defined so that `cmd.exe` is executed when the `notepad.exe` process is created on the system. [105, 103]

A powerful method to create WMI subscriptions for persistence is *PowerShell*. All three classes can be generated by submitting different parameters to the *New-CimInstance* or *Set-WmiInstance* cmdlets.

The commands can have a structure similar to the following:

```
$Filter = New-CimInstance -Namespace root/subscription  
-ClassName __EventFilter -Property $FilterArgs
```

```

$Consumer=New-CimInstance -Namespace root/subscription
-ClassName CommandLineEventConsumer -Property $ConsumerArgs

$FilterConsumerBinding = New-CimInstance -Namespace root/subscription
-ClassName __FilterToConsumerBinding
-Property $FilterToConsumerArgs

```

where the arguments to each command would be defined similarly as in the MOF example. The Set-WmiInstance would work almost identically with only changing the *-Property* flag for the *-Arguments* flag. [105, 101]

```

1  #PRAGMA NAMESPACE ("\\.\root\subscription")
2  instance of CommandLineEventConsumer as $Cons
3  {
4      Name = "Pentestlab";
5      RunInteractively=false;
6      CommandLineTemplate="cmd.exe";
7  };
8  instance of __EventFilter as $Filt
9  {
10     Name = "Pentestlab";
11     EventNamespace = "root\subscription";
12     Query ="SELECT * FROM __InstanceCreationEvent Within 3"
13           "Where TargetInstance Isa \"Win32_Process\" "
14           "And TargetInstance.Name = \"notepad.exe\" ";
15     QueryLanguage = "WQL";
16 };
17 instance of __FilterToConsumerBinding
18 {
19     Filter = $Filt;
20     Consumer = $Cons;
21 };

```

Figure 2.10: A sample MOF file content defined to achieve persistence as shown by *PentestLabs*.

[105]

Another method of utilizing powershell is to create instances of the three classes and then save them in the WMI repository via their own method:

```

$instanceFilter = ([wmi]class "\\.\root\subscription:__EventFilter")
                 .CreateInstance()

$instanceFilter.QueryLanguage = ...
$instanceFilter.Query = ...
$instanceFilter.Name = ...
$instanceFilter.EventNamespace = ...

$instanceFilter.Put()

```

2. EXPLORING PERSISTENCE TECHNIQUES

The above example describes the creation of *EventFilter*, the other two are created similarly by only setting their own relevant arguments corresponding with the *MOF* example. [106]

Persistence abusing WMI can also be created through a command line tool named *wmic*. It takes only three commands to complete basic persistence, that is, again, creating a filter, creating a consumer, and creating a bind: [105]

```
wmic /NAMESPACE:"\\root\subscription"
  PATH __EventFilter CREATE Name="<Name>", EventNameSpace="root\cimv2",
  QueryLanguage="WQL", Query="<Query>"
wmic /NAMESPACE:"\\root\subscription" PATH CommandLineEventConsumer
  CREATE Name="<name>", ExecutablePath="<path>",
  CommandLineTemplate="<template_path>"
wmic /NAMESPACE:"\\root\subscription" PATH __FilterToConsumerBinding
  CREATE Filter="__EventFilter.Name="<name_of_filter>",
  Consumer="CommandLineEventConsumer.Name="<name_of_consumer>"
```

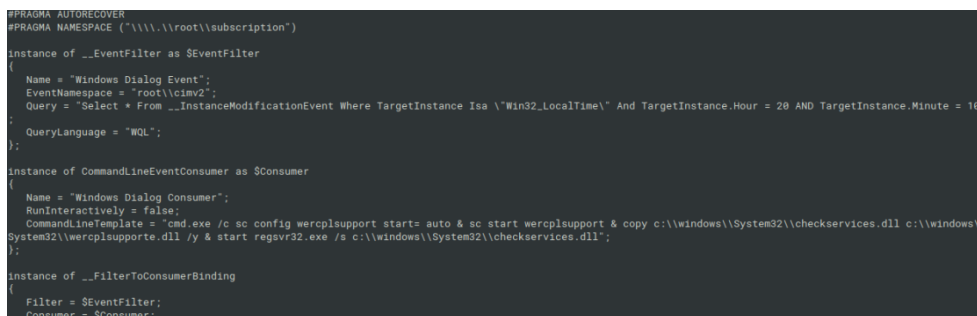
The same arguments as in *MOF* can be once again utilized.

2.2.3.2 Examples

There are not that many analyses available that would describe the exact method of establishing persistence using WMI Event Subscription for persistence, however, there are reports suggesting that the technique was used.

Blue Mockingbird Coin Miner

Persistence using WMI Event Subscription was discovered in a coin miner attributed to the Blue Mockingbird group. A sample analyzed by *Ladislav Bačo* from the *Lifars* company identified persistence gaining by defining and compiling a *MOF* file, as shown in figure 2.11.



```
#PRAGMA AUTORECOVER
#PRAGMA NAMESPACE ("\\\\.\\root\\subscription")

instance of __EventFilter as $EventFilter
{
  Name = "Windows Dialog Event";
  EventNameSpace = "root\\cimv2";
  Query = "Select * From __InstanceModificationEvent Where TargetInstance Isa `Win32_LocalTime` And TargetInstance.Hour = 20 AND TargetInstance.Minute = 10";
  QueryLanguage = "WQL";
};

instance of CommandLineEventConsumer as $Consumer
{
  Name = "Windows Dialog Consumer";
  RunInteractively = false;
  CommandLineTemplate = "cmd.exe /c sc config wercplsupport start= auto & sc start wercplsupport & copy c:\\windows\\System32\\checkserves.dll c:\\windows\\System32\\wercplsupporte.dll /y & start regsvr32.exe /s c:\\windows\\System32\\checkserves.dll";
};

instance of __FilterToConsumerBinding
{
  Filter = $EventFilter;
  Consumer = $Consumer;
```

Figure 2.11: Sample of the contents of a *MOF* file used by a malware sample belonging to the Mocking Bird group to gain persistence through WMI

. [108]

BADHATCH by FIN8

An analysis conducted by *Bitdefender* concludes that WMI Event Subscription was used as a persistence technique in a *BADHATCH* malware sample. They do not show exactly which commands or files were used to achieve that persistence, but were able to discover that the originator was an encrypted Powershell script and the Event Filter was defined by the following query:

```
SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE
  TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'
  AND TargetInstance.SystemUpTime >= 140
  AND TargetInstance.SystemUpTime < 240
```

That defines an action occurring when the system uptime reaches a 140 seconds after the boot up of the system. [109]

2.2.3.3 Detection

There is a native event log available in Windows that logs permanent WMI event subscription to *Microsoft-Windows-WMI-Activity/Operational* channel and that has ID *5861*. This event log can provide a lot of useful information, such as the namespace name and definitions of both the event filter and the event consumer. Based on this information, filters can be created so that only *CommandLineEventConsumer* and *ActiveScriptEventConsumer* are detected to avoid false positives. [110, 104]

What is more, command line activity can be detected using the event ID *4688*, checking for the presence of mentioned *wmic* tool with some of the likely used strings in the command (*\\root\subscription, CommandLineEventConsumer, ActiveScriptEventConsumer*) to detect the creation of an event consumer.

The same logic can be utilized with an event ID *4104* for Powershell script block logging, while hunting also for the above mentioned cmdlets (*New-CmiInstance, Set-WmiInstance*).

If *Sysmon* is enabled on the endpoint, event IDs *19, 20, 21* can also be utilized to monitor event filter, event consumer and event binding activities, respectively. [110, 104]

In addition, it is possible to detect the usage of *mofcomp.exe* in the command line using the event ID *4688*.

2.2.3.4 Artifacts

In terms of what to acquire after detecting a WMI subscription activity, the biggest object of interest is the content of the WMI repository (*%SYSTEM-ROOT%\System32\wbem\Repository*). There are tools that are able to parse

the repository and view all the present objects, including the possibly malicious definitions. There is, for example, the *flare-wmi* set of tools by Mandiant, or the *PyWMIPersistenceFinder* written in Python, which analyzes the *OBJECT.DATA* file from the repository to detect persistence. [103]

It can also be useful to collect any files identified inside the monitored commands as that will often be the applications set to start using the event consumer or related somehow to the activity. If a *MOF* file is identified, it is also beneficial to fetch it. As with the other techniques, the binary of malicious parent process would be a nice artifact for analysis.

2.2.4 BITS Jobs (T1197)

BITS (Background Intelligent Transfer Service) is a Windows service that allows applications to transfer files in the background without interrupting the user's work. It can transfer data, for example, to web servers via HTTP or file shares via SMB protocol. BITS is commonly used by updaters, messengers, or other applications operating in the background. The jobs are allowed to be present for a long time (90 days by default) and stay active even after a system reboot. That allows attackers to abuse it for persistence. However, it is even more commonly used for exfiltration, command-and-control communication, or to download payload. [111, 112, 113]

2.2.4.1 Principle

There are two native tools present in Windows that allow to create and manage BITS jobs. In the command line, *bitsadmin.exe* can be used. It is worth noting that to interact with *bitsadmin*, administrator privileges are required. The *Start-BitsTransfer* cmdlet can be used in Powershell to transfer a file via BITS.

To achieve persistence using *bitsadmin*, there is a flag */SetNotifyCmdLine*, which is run every time when the job is complete or ends with an error. The time period to attempt to perform the job can be set with the */SetMinRetry-Delay* flag. A typical sequence of steps performed by an adversary to achieve persistence through *bitsadmin* would look as follows: [111, 112, 114, 116]

1. Create a BITS job:
`bitsadmin /create <jobname>`
2. Define from which remote address to which location should the job attempt to download (the remote address can be anything, as the goal is for the job to fail and attempt the file transfer again and again):
`bitsadmin /addfile <jobname> <any_url> <any_real_file>`

3. Define the command or a binary set to run every time the job is attempted:
`bitsadmin /SetNotifyCmdLine <jobname> <command> NUL`
4. Set the recurrence period for the job:
`bitsadmin /SetMinRetryDelay <jobname> <seconds>`
5. Activate the job:
`bitsadmin /resume <jobname>`

In Powershell, it is enough to use one command:

```
Start-BitsTransfer -DisplayName <jobname> -NotifyCmdLine  
<command_to_run> -NotifyFlags JobError -RetryTimeout 60  
-RetryInterval 60 -Source <random_source> -Destination  
<existing_path> -Asynchronous
```

This command sets the retry timeout and interval after error to 60 minutes. [118]

2.2.4.2 Examples

BITS jobs abuse was detected in many malware families (e.g. QakBot) for performing various tactics. The typical usage for persistence is best visible in the UBoatRAT trojan. [113]

UBoatRAT

As revealed in an analysis by Palo Alto Networks, UBoatRAT utilizes the previously shown *bitsadmin* sequence of commands to achieve persistence (figure 2.12).

```
bitsadmin /create dlf2g34  
bitsadmin /addfile dlf2g34 c:\windows\system32\net.exe %temp%\sys.log  
bitsadmin /SetNotifyCmdLine dlf2g34 "c:\programdata\svchost.exe" ""  
bitsadmin /Resume dlf2g34  
Del %0
```

Figure 2.12: Series of *bitsadmin* commands used by *UBoatRAT* to achieve persistence.

[115]

2.2.4.3 Detection

Although there are operational Windows event logs for BITS jobs specifically, they lack all the information needed for detecting persistence in particular. To recognize persistence attempts specifically in the usage of BITS jobs, the best information come from item 3 of 2.2.4.1, which sets up the notification command. From there, it is even potentially possible to recover the malicious file set to run. During experimenting with the event logs, information about this particular activity was not found there. If detection was based only on the event logs, it would be hard to distinguish other tactics and legitimate BITS use from persistence.

The most effective way is to utilize event ID *4688* and hunt for usage of *bitsadmin* with relevant flags (*/SetNotifyCmdLine*, */SetMinRetryDelay*) in the command line. To detect jobs created in Powershell, script block logs can be searched for presence of the *Set-BitsTransfer* cmdlet with flags *NotificationCommandLine* or *MinimumRetryDelay*.

2.2.4.4 Artifacts

Although the BITS operational logs are not used for detection, it is an artifact that should be available for additional investigation. A true forensic artifact specific for BITS and present on the disk are state files. These are two files located at *C:\ProgramData\Microsoft\Network\Downloader* named *qmgr0.dat* and *qmgr1.dat*. There are tools that can recover information about BITS jobs from these files, for example Bits-Parser. [112, 117]

2.2.5 Windows Service (T1543.003)

Windows Services is a utility that allows users and administrators to run executables in the long term without the need to manually start them. They can be started automatically on system startup, which is a characteristic that is very attractive for attackers looking for methods of gaining persistence. Services can be started and stopped or just left running in the background without showing any interface. The reason why it may not be as popular as, for example, scheduled tasks or registry run keys, is that to create a service, administrator privileges are required. However, when an attacker already has those, services can be misused to elevate privileges as they typically run with system privileges. [119, 120]

2.2.5.1 Principle

There are multiple methods how to add a new service as an attacker and have it launched after each system boot. A common way is to register and start service manually utilizing command line:

```
sc create <service_name> binpath="<binary_to_run>" start="auto"  
  obj="LocalSystem"  
sc start <servicename>
```

sc.exe is the utility allowing management of services. The parameter *create* registers the service. Thanks to the *start="auto"* parameter, the service is started on each system boot. Attackers can also utilize the *config* parameter instead of *create* to modify an existing service.

The same goal can be achieved via Powershell utilizing the *New-Service* cmdlet:

```
New-Service -Name <service_name> -BinaryPathName <binary_to_run>  
  -Description <description> -StartupType Automatic  
Start-Service -Name "<service_name>"
```

It is necessary to start the services by the *start* parameter when using *sc* or by the *Start-Service* cmdlet. [121, 122, 123]

The configuration of the services is saved into the following registry key: *HKLM\SYSTEM\CurrentControlSet\Services*, so with administrator privileges, it is possible to edit it and add a new service directly. However, to activate it, a reboot is needed. WINAPI functions that can create Windows services are also available. It is therefore possible to create Windows tasks without command line usage, which needs to be taken into consideration when designing detections. [120]

2.2.5.2 Examples

Blue Mockingbird Coin Miner

As detected by *RedCanary*, aside from abusing WMI, Blue Mockingbird is also capable of establishing persistence by creating a Windows service via *sc*.

```
sc create 8995 binPath= "cmd /c sc config wercplsupport start= auto & sc start wercplsupport & copy  
c:\windows\System32\8995.dll c:\windows\System32\wercplsupporte.dll /y & regsvr32.exe /s c:\windows\System32\8995.dll"  
type= share start= auto error= ignore DisplayName= 8995
```

Figure 2.13: Utilization of *sc* to create a persistent Windows service by a Blue Mockingbird coin miner sample as detected by RedCanary.

[124]

Ragnar Locker Ransomware

As reported by Sophos, Ragnar Locker ransomware also uses a Windows service to persistently run necessary components to support its VirtualBox installation.

```
%binapp%\VBoxSVC.exe /reregserver
regsvr32 /S "%binpath%\VboxC.dll"
rundll32 "%binpath%\VBoxRT.dll,RTR3Init"
sc create VBoxDRV binpath= "%binpath%\drivers\VboxDrv.sys" type= kernel start= auto error=
normal displayname= PortableVBoxDRV
sc start VBoxDRV
```

Figure 2.14: Utilization of `sc` to create a persistent Windows service by a Ragnar Locker ransomware sample as detected by Sophos.

[125]

2.2.5.3 Detection

There are two native Windows events that are logged when a new service is created on the system, a security event log with ID *4967* and a system log with ID *7045*. Additional detections can leverage known command line strings, such as the `sc` command in combination with the `create`, `config` or `binpath` arguments. With the Powershell script logging, it is possible to monitor for the `New-Service` cmdlet with the `BinaryPathName` parameter.

Detection of a new key inside this particular location:

`HKLM\SYSTEM\CurrentControlSet\Services\` or a modification of any existing subkeys is possible. However, it is necessary to consider the potential false positive rate and perform proper baselining as this key is commonly changed by legitimate applications. [120]

2.2.5.4 Artifacts

Again, all paths present in the detected events are interesting to recover. It is also helpful to obtain the configuration of the newly set service. For a thorough investigation, it can be useful to acquire all the subkeys of `HKLM\SYSTEM\CurrentControlSet\Services\`, however, this can typically be hundreds of entries, so the main focus would be on the newly added or modified ones.

Laboratory Environment

After understanding the persistence techniques theoretically, the goal is to take steps to utilize this knowledge in a practical environment. It is necessary to define which technologies are to be used and how they are supposed to work together. There are three main components that the laboratory consists of: log management, monitored endpoint device, and a solution for obtaining interesting artifact remotely.

3.1 Cloud Environment Design

The laboratory components will be placed in the cloud and consist of multiple virtual machines. The chosen cloud platform is **Microsoft Azure** as it has an academic account option with sufficient free credit for this type of laboratory environment. The same functionality could, however, be implemented on other cloud platforms as well as in a local environment.

For purposes of testing the persistence techniques, one Windows endpoint will be sufficient. This endpoint will simulate a workstation of a general user that is susceptible to malware infection. The endpoint will be monitored and will send the appropriate logs to a log management solution. There, rules will be implemented so that selected persistence techniques are detected, making use of the analysis in the previous sections. In addition, additional artifacts will be collected to improve the alert analysis capabilities by a special tool from the endpoint.

3.1.1 Log Management

There are multiple SIEM and log management options available. Many products are proprietary with expensive licenses and also require a lot of resources. For a solution which is hosted in Azure cloud with limited resources, it is convenient to choose an appropriate tool. Also, the tool should be easy to deploy and current trends should be taken into account. The chosen log management

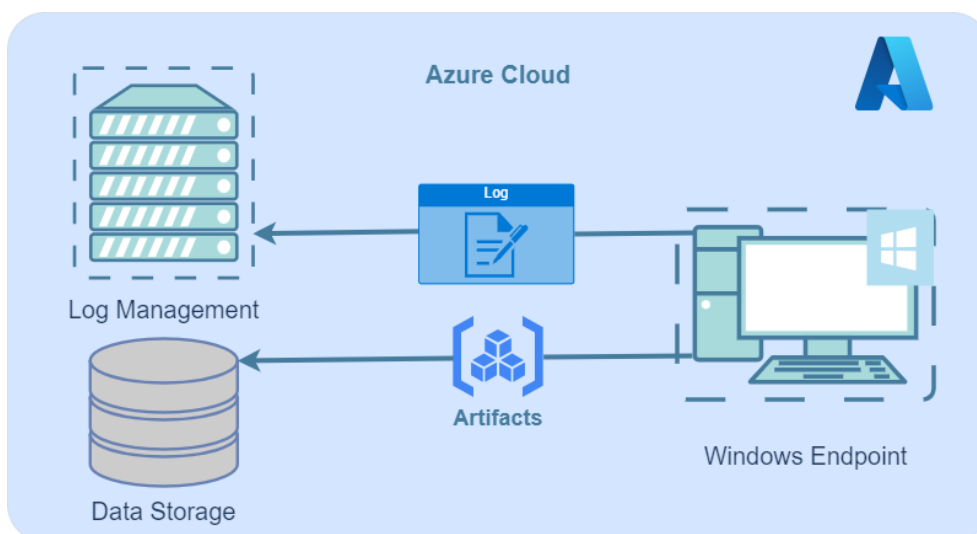


Figure 3.1: High level scheme of laboratory in Azure Cloud.

solution is **Splunk**. It is known to be easily deployable in the cloud and also, according to *6sense*, it has a market share of more than 50%. [85]

3.1.1.1 Splunk Components

Splunk is a platform that consists of several components that can be connected together based on its deployment architecture, which is usually determined by the expected amount of data, number of monitored machines and other requirements: [86, 87]

- **Splunk Indexer** - An indexer is a component taking care of data storage. It processes and stores incoming data and turns them into events that are ready for searching. There can be multiple indexers in place, depending on the deployment architecture.
- **Splunk Search Head** - This is the main component which a user comes into contact with. It is a Splunk instance with a web interface through which it is possible to perform searches in the indexed data. When multiple indexers are deployed, the search head distributes searches between them. A search head can also have its own indexes, so it can function even without usage of dedicated indexers. That is suitable for an environment with only a small amount of data stored.
- **Splunk Forwarder** - Forwarder is a component which is typically installed on monitored devices. It is an application that sends selected information to indexers. There are two types of forwarders. A *universal forwarder* is more simple, it does not modify or index the data in

any way, it just sends it raw to a selected indexer. A *heavy forwarder* performs also parsing and indexing of the data before sending it to the indexer. This demands more time, but in a larger environment the data parsed can be used for effective routing.

- **Splunk Deployment Server** - When multiple components are deployed, this server handles their management. It can distribute configurations, Splunk apps and updates to various types of instances under its management. This can be done in bulk for different resource groups.

3.1.1.2 Architecture of Splunk in the Lab

The official Splunk documentation states 4 main types of distributed⁵ deployment based on the size of the environment. The smallest is labeled as *Departmental*. As illustrated in figure 3.2, the *search head* and *indexer* both run together on a single splunk instance while receiving logs from up to 10 forwarders. This setup is suitable for a maximum of 10 users. [89]

The *departmental* deployment scheme will be used in our laboratory setup as it is only necessary to monitor one endpoint machine and therefore collect data from only one forwarder. If necessary, additional machines can be created. This setup does not require usage of a deployment server as it is only necessary to configure one forwarder. However, the main Splunk instance can be used as a deployment server if needed.

The main focus of laboratory work is the detection of persistence techniques. The rules which are implemented into the Splunk environment can, however, be used in any infrastructure and size. The *departmental* deployment was chosen for laboratory purposes, but the same rules can be implemented even in the biggest, *large enterprise*, deployment scheme.

3.1.2 Remote Artifact Collection

It is necessary to think ahead while designing the laboratory and to tackle the topic of remote artifact collection. One of the goals of this work is to enrich the detected alerts with additional artifacts acquired from the monitored endpoint. In the basic version of Splunk, it is possible to set actions after alerts, however running scripts on the particular machine which generated the alert is not supported with the use of universal forwarder. Therefore, it is necessary to look into a different solution.

For these purposes, there is a suitable tool called **Google Rapid Response (GRR)**. In basic usage, there is one server that runs the GRR and there are agents deployed to endpoints. GRR is a tool that allows live forensics

⁵There is also a possibility of a single instance deployment which consists of a single machine acting as the only log source. This type is suitable for educational or experimental purposes, practical solutions run on distributed or even clustered deployment. [88]

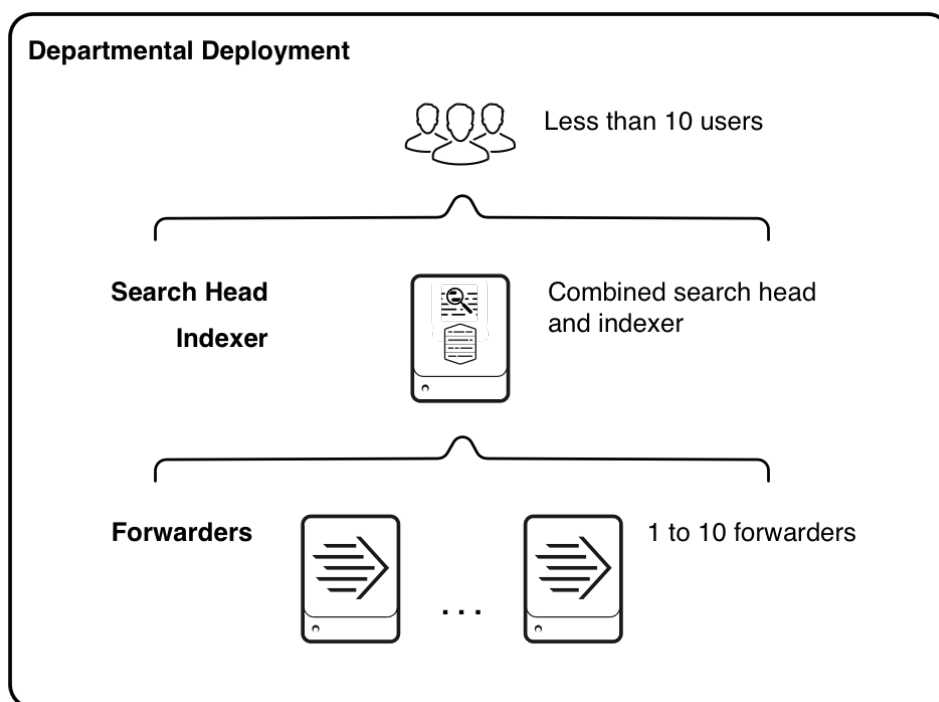


Figure 3.2: Departmental deployment scheme of Splunk.

[89]

with a set of utilities. It enables the analyst to virtually browse the filesystem and Windows registry. It can also provide information about memory. Most importantly, it can collect selected forensic artifacts or any desired file from the filesystem remotely and on demand. [95]

3.1.3 Windows Endpoint Machine

This endpoint will serve as the only device that simulates a user workstation. It will be a Windows 10 virtual machine deployed in Azure. To get logs into splunk, it will be necessary to install a *forwarder*. Most likely, it will be necessary to configure the machine and the forwarder for the purposes of obtaining correct and relevant logs. It is also possible that additional tools will be installed for the same purpose. That will be discussed in the chapter about implementation.

3.2 Setting up Azure Environment

Azure is the selected cloud platform for deployment of the virtual machines. For basic functionality, three virtual machines will be set up - one playing

the role of a user workstation and the other ones will serve as the analysis platform with a Splunk Enterprise instance and a GRR server. For the reason of scalability and to maintain good practices, the analysis environment and the endpoint are placed into their separate virtual networks. Also, each of these machines has its own networking rules which allow only necessary connections.

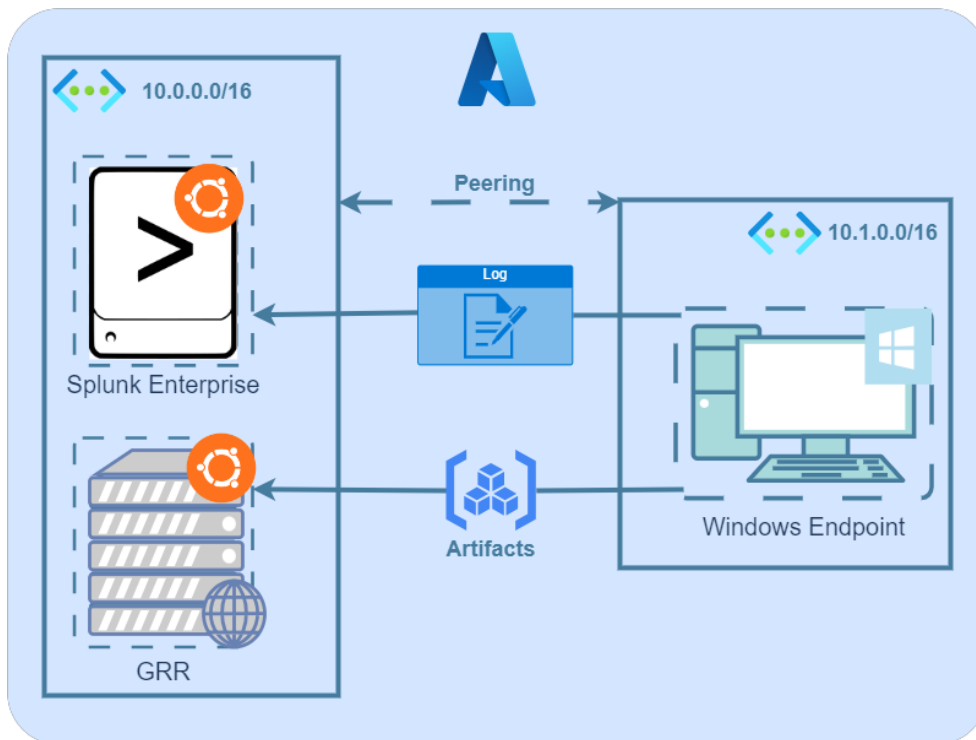


Figure 3.3: Scheme of the laboratory environment in Azure with concrete technologies and subnets highlighted.

3.2.1 Splunk Enterprise Instance

The first deployed machine is dedicated to host the Splunk Enterprise instance. Selected operating system is *Ubuntu 22.04 LTS* server version (without GUI). As of performance, *Standard D2as v4* version with 2 CPUs and 8 GiB of RAM was selected as it should be enough to satisfy the needs of the Splunk instance and at the same time, its credit consumption is still reasonable. The name of the machine is *splunk-server*. As illustrated in figure 4.4, it is a part of a virtual subnet *10.0.0.0/16* with its own IP address *10.0.0.4*. When it comes to the network rules, it is only necessary to open port *22* for SSH connections.

Installing the Splunk instance itself is a straightforward process. A Splunk package can be downloaded with the *wget* command, the full path is available on the Splunk website after creating a profile. There, it is possible to

manage licences. For these purposes, the developer version is sufficient. After downloading the package, it can be installed using the *apt install* command. Splunk is automatically installed into the */opt/splunk* directory. After that, the Splunk instance can be started, an administrator profile is created and the web interface is available by default on port *8000*. The web interface can be accessed on the public IP address of the *splunk-server* virtual machine. It is only necessary to add an inbound network rule to port *8000* to allow inbound access to the web interface. Then it is convenient to register Splunk as a service and schedule it to run on system startup.

3.2.2 Google Rapid Response Instance

The GRR instance runs on an equivalent *Ubuntu 22.04 LTS* server. Its hostname is **grr-server** with the IP address being *10.0.0.5*. Aside from the management SSH port, it is necessary to open also the port *8000* for serving the web user interface. For communication with the clients, GRR uses a special protocol called *Fleetspeak*. The service listens on the port *4443*, so this port was also enabled for the endpoint VM. Note that in a production environment, the web application would be served through an Apache or Nginx proxy to ensure usage of HTTPS. The Fleetspeak communication is encrypted by default.

The installation itself seems to be pretty straightforward, with the only prerequisite being setting up a *MySQL* database. The installation of the GRR requires some information about hostnames and IP addresses that it is going to be served through. However, after entering the public address of the **grr-server** into all the relevant prompts in the installation, there was an issue running Fleetspeak service. After configuring Fleetspeak with the local address, the service was running properly, however, another problem occurred while trying to connect the client from the Windows endpoint. This was resolved by changing the local address of the **grr-server** to its public address in the fleetspeak client configuration file (*C:\Windows\System32\GRR\3.4.7.1\Fleetspeak*). This results in the client connecting to the server with its public IP address. After many experiments, no other working configuration was found.

3.2.3 Windows Endpoint Machine

Requirements for this machine are not very high. The OS deployed is *Windows 10 22H2* to simulate a common environment of a user workstation. Performance setting is *Standard D2as v4*, which offers 2 CPUs and 8 GiB of RAM, which is more than sufficient for the laboratory purposes. The endpoint name is *endpoint-win1* and the name of the simulated user is *endpointuser1*. As illustrated in figure 4.4, it is a part of a virtual subnet *10.1.0.0/16* with its own IP address *10.1.0.4*. To connect to the machine, RDP can be used.

Therefore, by default, there is a network rule to leave the RDP port *3389* open for incoming connections.

3.2.4 Setting up Log Flow

After successfully deploying the virtual machines, the next step is to establish log flow from the endpoint to Splunk. As discussed in previous section, a minimalistic solution that matches the *Departmental* architecture will be used. The indexer and search head are served by the Splunk instance deployed on the *splunk-server*. There are two options for how to employ the forwarder to provide logs from the endpoint. Either it is possible to utilize a *universal forwarder* which has to be installed directly on the machine, or by using *native WMI logging*. The latter is more suitable for an Active Directory environment as there has to be a forwarder running as a domain user. For purposes of this laboratory, it is sufficient to install a universal forwarder directly on the endpoint, which is also recommended by Splunk. [90]

The installer is publicly available on the Splunk website. During installation, it is only necessary to select which Windows logs are to be collected (*application*, *system* and *security* logs were selected to be forwarded in Splunk indexer, however the log selection will be conducted continually for effective detection of individual techniques) and enter the IP addresses of the *deployment server* (omitted in this small architecture) and the *indexer*. In this case, the logs should be sent to *10.0.0.4* with the default port of *9997*.

For this connection to work, it is necessary to make additional configuration changes both in Azure and in the Splunk Instance:

- In order to send data from the virtual subnet of the endpoint to the virtual subnet of the Splunk instance, it is necessary to setup *peering* in Azure between them. After that, the two subnets are able to communicate with each other. In a live, non-laboratory scenario, the subnets would be connected through a network element (typically a router or a firewall).
- Set up a new receiving interface inside the Splunk Instance. That is done by adding a configuration for data receiving through the web interface. The port used is *9997*.
- A new network rule has to be introduced on the *splunk-server*, so that the connection from the forwarder can be established. It is only necessary to set an incoming allow rule from *10.1.0.4* to port *9997*.

At this point, some logs start to flow from the Windows machine and it is possible to explore them in the Splunk Search app. It is not necessary to set up anything additional into Splunk for basic functionality as the event logs have XML format. By default, Splunk is able to parse the logs and searches can be conducted across the data and the individual fields.

Implementing Detection Mechanisms

After setting up the environment and ensuring that logs can flow from the endpoint to the log management, it is necessary to implement detection mechanisms for each technique, utilizing the information discussed in their analysis. To achieve that, it will be necessary to configure the endpoint machine accordingly and to set up detection rules in the log management solution. Additional configurations and scripting will also be carried out in order to acquire artifacts related to the alerts using the Google Rapid Response platform.

For each technique, additional configuration, concrete detection, and artifact acquisition implementation will be described as well as manual testing. In this part, real malware is not run yet, the techniques will be tested only by manual actions on the endpoint simulating what a malware could generate in connection with the particular persistence attempt. All the custom files and scripts created during the implementation will be added to attachments.

4.1 Splunk Alerts

To detect potential malicious activity, it is necessary to set up alerts. In Splunk, alerts have a simple mechanism. Alerting in Splunk is based on search queries. The analyst can set up and run a search query and then analyze the results. In addition, this search can be saved as an alert. The alert is triggered each time the search query returns any results. There are several settings that can be made for the alert.

Alerts can be triggered in real-time, however, as Splunk is more of a log management solution without a correlation engine, it is demanding on resources and certain search commands are not supported. The alert can also be triggered in selected intervals (e.g. every 5 minutes). In that case, the search is conducted each 5 minutes, and if there are any results, an alert is

triggered.

Severity can be assigned to the alert, as well as some throttling conditions, for example, suppressing alert triggering for some time period after a trigger.

Alert actions can also be defined, which will be useful for automated acquisition of data from the endpoint machine. Custom scripts can be run after the alert is triggered with the particular variables from the search results.

Detailed overview of alert functions and options is available in the Splunk documentation. [93]

In the basic version of Splunk, there is a very limited possibility of alert management. Severity can be assigned, however, alert workflow cannot be natively implemented. There are Splunk add-ons that help with alert management (e.g., Splunk Enterprise Security is a popular solution for businesses). Another common option is to send triggered alerts to a SOAR or a ticketing software for further management. In this laboratory solution, the native alert overview will suffice.

4.2 Utilization of GRR

Once the Google Rapid Response platform is deployed, it is necessary to connect it with Splunk. The goal is to fetch relevant artifacts automatically when a potential malicious activity is detected, which will be done automatically using GRR API and its Python client invoked when a Splunk alert is activated.

4.2.1 GRR and Splunk Integration

Splunk allows to create a custom action after an alert is triggered. The process is not as straightforward as one would expect, as it is necessary to create a Splunk application. There is also an obsolete option of running only a single script after an alert is triggered. In this solution, the more complex solution is employed. It also has some advantages, for example, easier passing of parameters into the automated alert action.

Firstly, it is necessary to create a Splunk app using the user interface. The next steps are performed directly on the server. There are some configuration files needed to be set up for the alert action to be registered, as well as creating a simple HTML user interface in order to be able to display the alert action in alert settings and allowing user to pass desired parameters.

The main alert action logic is launched in a Python script which is invoked as the alert is triggered. Naturally, all the event data that resulted in the alert are passed to the script. Additionally, there is a possibility to hand over additional static parameters (defined in configuration files directly on the server) or dynamic parameters of various types that are set when the alert is created. In this solution, only one parameter will suffice. A string with the alert type will tie each alert with its corresponding alert action performed by

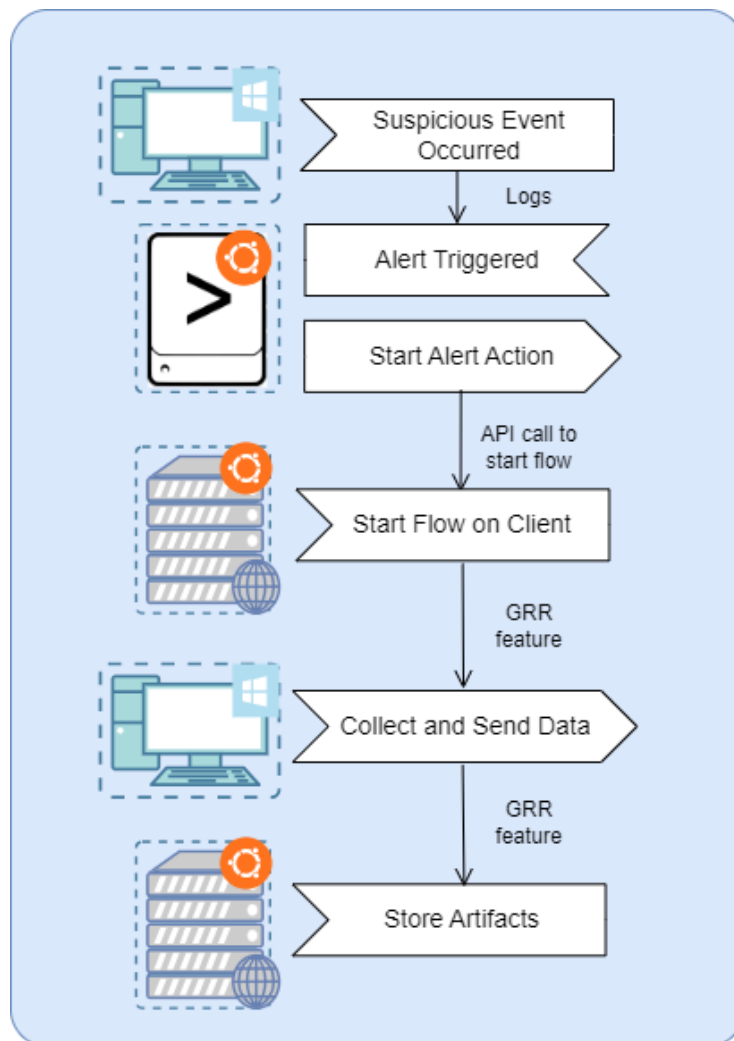


Figure 4.1: Design of the alert workflow between Splunk, GRR and Windows endpoint machine.

the script. Each alert has its own reactive action with different data being acquired.

For this purpose, a simple naming convention is introduced in the format of $\langle MITRE\ ID \rangle . \langle Abbreviation \rangle$, since one technique can have multiple alerts. For example, to detect a change in the startup folder, the alert will have the following name: *T1547.001.STF: Persistence attempt via Startup Folder detected*, where the **T1547.001.STF** part plays the role of the identification string for the alert action.

To correctly configure the custom alert action, specific files must be defined in specific folders. That is done accordingly to the Splunk documentation: [96]

4. IMPLEMENTING DETECTION MECHANISMS



When triggered	>	 Add to Triggered Alerts	Remove
	∨	 Handle with GRR	Remove
		Alert category	<input type="text" value="T1547.001.STF"/>

Figure 4.2: Design of the user interface of custom alert action to tie the alert with GRR action. One parameter is passed to define which action should be taken through GRR.

- A new directory with the application name is to be created inside the `/opt/splunk/etc/apps/` directory. This application will be called `grr_handling`.
- A configuration file has to be defined: `.../apps/grr_handling/default/alert_actions.conf`. The application name and the description is defined there, also the format in which the alert data should be passed into the script (JSON was selected). Furthermore, static parameters are defined there, here being only username and password to use with the GRR API.
- The `.../apps/grr_handling/metadata/default.meta` file has to be added so that the alert action is visible in the Splunk UI.
- The file `.../apps/grr_handling/default/data/ui/alerts/grr_handling.html` defines the interface as seen in figure 4.2. It acts as a form and in this solution, only the textbox is there to serve for entering the alert identifier.
- The dynamic parameters entered through the UI have to be defined inside `.../apps/grr_handling/README/savedsearches.conf.spec` along with the data type of each parameter. Here, it is only one string parameter carrying the alert category as described above.
- The python script `.../apps/grr_handling/bin/grr_handling.py` is the one invoked after the alert is triggered. It has to have the same name as the application name configured in `alert_actions.conf`. In the same directory, other helper scripts are also stored for the functionality itself.

All the files are available in the appendix.

One important issue while running custom alert actions using Python is modules that are not pre-installed in Splunk itself. Splunk is installed with separate Python language, in this case it is Python version 3.7. The automatically invoked script is therefore run using this installation. In order to

interact with the GRR API, a custom Python module was made available by GRR authors. This module can be installed even into the Splunk installation of Python. However, its dependencies require a higher version of Python (3.8). That would not be a problem if Splunk was using the system version of Python (3.10). Splunk does not recommend trying to upgrade its internal Python distribution, so it was necessary to use a workaround. Inside *alert_action.py*, the system python was invoked and other script with the desired functionality (*alert_action_original.py*) was run using the same parameters that were passed through from the alert:

```
for envvar in ("PYTHONPATH", "LD_LIBRARY_PATH"):
    if envvar in os.environ:
        del os.environ[envvar]

python_executable="/usr/bin/python3"
real_script="/opt/splunk/etc/apps/grr_handling/bin/\
---grr_handling_original.py"

os.execv(python_executable, [python_executable, real_script]\
        + sys.argv[1:])
```

4.2.2 Acquiring Files through GRR API

The *Google Rapid Response* platform contains one main feature through which an action can be performed on the monitored endpoint, which is called *flows*. Multiple actions can be carried out using one *flow*, however, it can be only deployed on one machine, which is sufficient for our purposes. Still, invoking *flows* through multiple machines is allowed by GRR by a feature called *hunts*.

It is possible to utilize a *flow* for many actions, including file acquisition or process memory dump, which are very useful for our purposes. Other interesting actions are available, such as execute a selected binary or a Python script on the endpoint, or collect file hashes. [95]

In this solution, a flow will be invoked after a specific alert action. Actions will be carried out based on the particular alert. Mostly, some selected files will be fetched from the endpoint, or a process memory dump will be conducted. The *flow* results will then be visible in the GRR interface. That has a slight disadvantage, as the individual flows cannot be named or tagged, so the potential analyst has to identify the flow only by associating alert trigger time in Splunk and in GRR. However, the flows are organized under their corresponding endpoints and in a well tuned environment, there should not be many flows invoked automatically on one endpoint machine.

4.3 Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)

A straightforward way to detect this persistence technique is to utilize *Windows Event Logs* as described in section 2.2.1.3. The basic event logs are already flowing into Splunk, so it is not necessary to install another tool on the endpoint. Of course, it can be beneficial to utilize a supporting tool for better logging configuration or data. However, if the detection can be done effectively by using native capabilities, time and resources are saved.

4.3.1 Additional Endpoint Configuration

In this case, as mentioned in the design section, the logs that are applicable for the detection are not enabled by default. Therefore, it is necessary to update the policy settings and enable this audit log. As mentioned, the log useful for the detection of unwanted registry manipulation with ID *4657* is a part of the **Security** audit logs. It falls into the *Object Access* category and *Audit Registry* subcategory. To enable only this subcategory, it is necessary to edit *Advanced Security Policy Configuration*, which is accessible via the *secpol.msc* Windows tool. Both *success* and *failure* events can be turned on, as the information about failed registry changes can be also valuable (e.g., if the attacker does not have enough privileges to modify the registry key). [91]

There is one additional configuration to be made. Even when the *Audit Registry* group policy is enabled, it does not monitor changes in all registry keys. As stated in the Windows documentation, the particular logged events are generated only for registry keys where the SACL is set accordingly. The majority of registry keys are therefore still not monitored, including those relevant for targeted detection. [69, 91]

There are two possibilities for how to set the SACL. Either individually for each relevant key that we want to monitor or globally across all registry keys. Unsurprisingly, this has an impact on the volume of logs sent from the endpoints. Setting the SACL globally can have a large impact on storage size, especially in a larger environment. The selected option needs to be decided during implementation, there is a possibility to save storage and have smaller amount of logs counterbalanced by the fact that it is necessary to set the SACL for each registry key that is intended to be monitored. On the other hand, the SACL can be set globally, which saves configuration complexity, but the amount of logs is significantly larger and not that selective. However, it has the advantage that additional detections can be created in the log management or SIEM alone without changing configurations of the machines. For the purposes of this work, it came to choosing the individual configuration for each key as with the global settings, the amount of (for this use-case) unnecessary logs was very high and resulted in throughput problems on the Splunk server. A

4.3. Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)

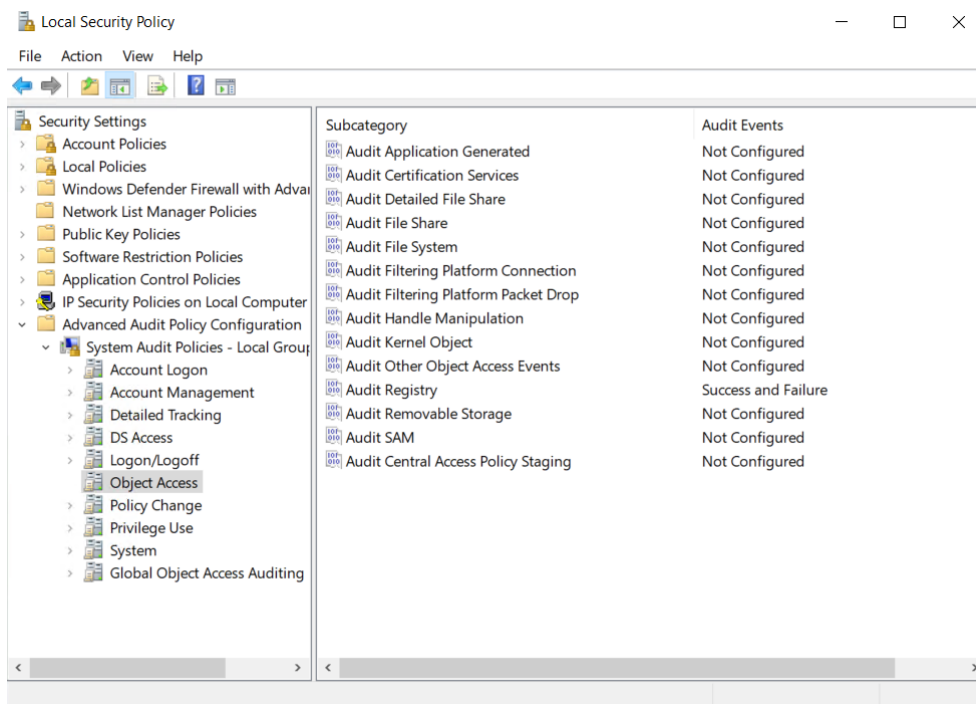


Figure 4.3: Enabling the *audit registry* policy using the Local Security Policy editor (*secpol.msc*).

Powershell script was created in order to set the appropriate SACL for all selected registry keys (*set-registry-sacl.ps1*).

The global configuration can be done similarly in the *Advanced Security Policy Configuration* under the *Global Object Access Auditing* tab. It is then necessary to select a security principle for tracking, that is going to be ***Everyone***. For tracking event ID *4657*, it is enough to select the ***Set Value*** activity to monitor according to the documentation. Both *Success* and *Failure* actions will be monitored. Other categories would generate additional, now unnecessary logs. [69]

This configuration prepares the capabilities for detecting the modification of all registry values identified in the analysis in section 2.2.1.1.

More configuration is needed to detect changes in the startup folders. As discussed in the analysis section, the event with ID *4656* will be used. Now, it is necessary to monitor two specific directories. Previously enabled SACL for the registry will not be useful. Instead, individual SACL for monitoring new file or folder creation will be set for these two directories. In order for the log to be active, it is necessary to enable *Audit File System* and *Audit Handle Manipulation* under the *Object Access* advanced audit policies tab.

Another monitoring method discussed during the design was **command**

4. IMPLEMENTING DETECTION MECHANISMS

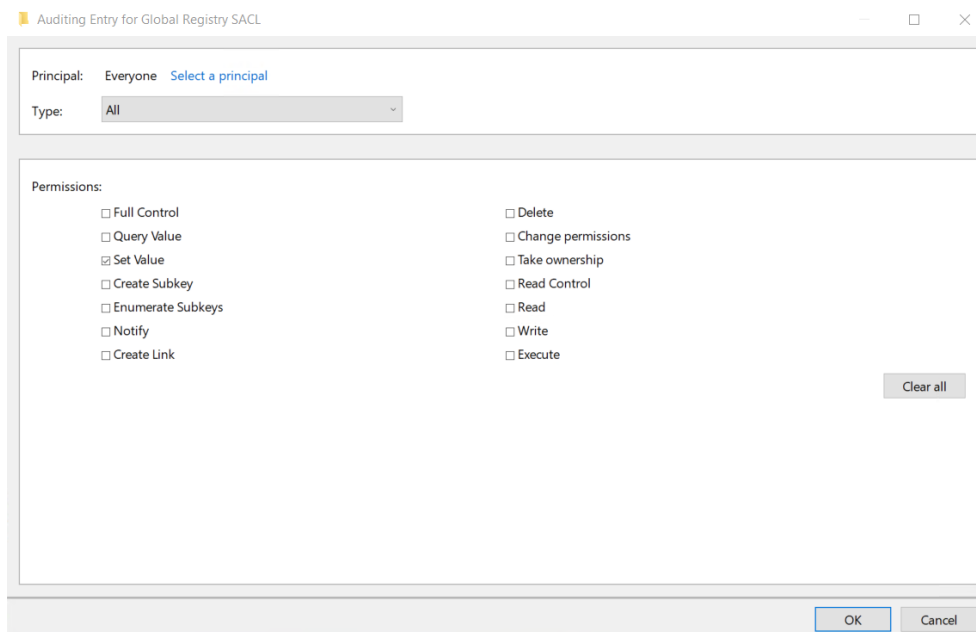


Figure 4.4: Selecting the activities to monitor in connection with global registry SACL policy.

line detection. Here, the event ID *4688* will be utilized. Again, policies need to be set to enable the log, this time under *Detailed Tracking* and *Audit Process Creation*. This event monitors the creation of a process. For detection and visibility purposes, it is desirable to see the exact command that was launched even with its parameters. That is also a subject of manual additional configuration. It needs to be enabled under *Computer Configuration/Administrative Templates/System/Audit Process Creation* tab in the GPO editor as shown in figure 4.6. [92]

The last configuration is also made to monitor the commands. Malicious commands can sometimes be run from Powershell scripts. If that is the case, the malicious command itself is not seen in the event log *4688*. Therefore, *Powershell Block Logging* will be enabled so that the script content is logged. This is also done through group policy settings. It will be enabled under *Computer Configuration/Administrative Templates/Windows Components/Windows PowerShell*. When logging is enabled, the contents of the script are logged in the event ID *4104*. [94]

However, these logs are not logged into the default *Security*, *System* or *Application* EVTX files. Therefore, it is necessary to configure the Splunk forwarder so that the file containing the newly enabled logs is present. In this case, it is *Microsoft-Windows-PowerShell%4Operational.evtx*. To monitor this file, it is necessary to create an *inputs.conf* file inside the *etc* folder

4.3. Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)

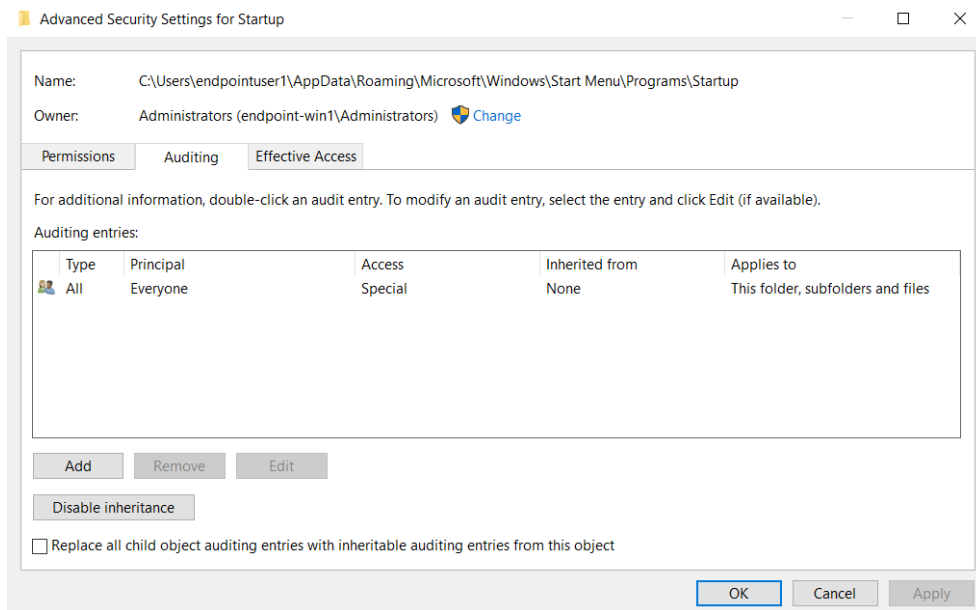


Figure 4.5: SACL policy to enable monitoring of the two default startup folders.

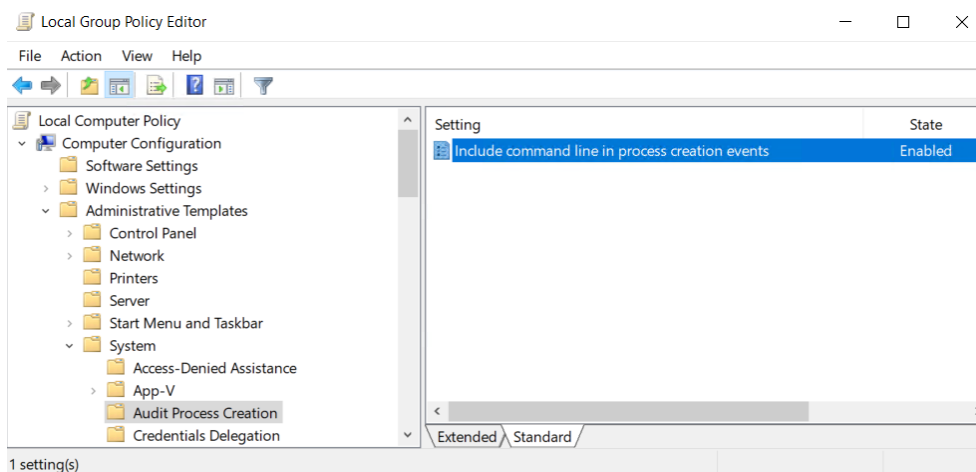


Figure 4.6: Policy setting to enable logging of whole commands in command line for event ID 4688.

of the Splunk forwarder installation. There, the configuration to monitor this particular log file is enabled by defining the concrete log file and setting disabled to 0: [90]

```
[WinEventLog://Microsoft-Windows-PowerShell/Operational]  
disabled = 0
```

4. IMPLEMENTING DETECTION MECHANISMS

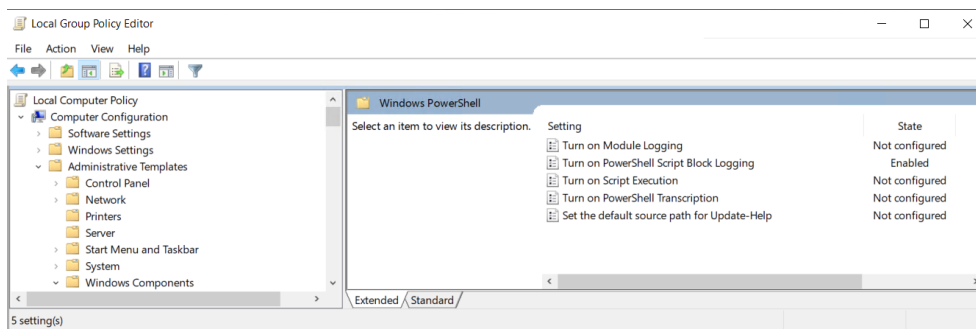


Figure 4.7: Policy setting to enable PowerShell script block logging in order to invoke event ID 4104.

4.3.2 Detection and Alerting

After configuring everything to monitor registry modification, it is necessary to create the detection rules in Splunk. Four rules (alerts) will be implemented for this technique. One of them alerts on an attempt to tamper with the contents of the startup folder, which is quite straightforward. However, while designing rules for the second part of the technique which abuses registry keys, it is also necessary to plan ahead for alert post-processing and taking into account Splunk alert behavior. To better extract information from the logs for the purposes of acquisition of additional analytic material, the detection is not designed as one large rule. Three separate rules are created, each detecting different event ID.

T1547.001.STF: Persistence attempt via Startup Folder detected

To detect the startup folder abuse, event ID 4656 is used while hunting for one of the identified paths inside the *Object Name* field.

```
index=main EventCode=4656 (Object_Name="*Users\\*\\AppData\\  
Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\  
*" OR Object_Name="*\\ProgramData\\Microsoft\\  
Windows\\Start Menu\\Programs\\StartUp\\*")
```

In connection with this event, it is desired to acquire all the files present in the startup folder for immediate analysis. The *GRR File Finder* flow is used to fetch the file which is found under the *Object Name* property of the event data. It is necessary to set up the *DOWNLOAD* flag so that the contents of the data are fetched as the default option of the *FileFinder* flow is *STATS*, which retrieves only the metadata of the selected file.

T1547.001.GEN: Persistence attempt via Registry Run Keys detected

To monitor unwanted manipulation with the identified keys generally, an alert monitoring events with ID *4657* is introduced. The query looks for corresponding strings in the *Object Name* field using a regular expression:

```
index=main EventCode=4657 Operation_Type = "*created" |
  regex Object_Name="(?!i)(^\\\\\\REGISTRY\\\\\\(MACHINE|USER\\\\
  [^\\\\\\]+)\\\\\\SOFTWARE\\\\\\Microsoft\\\\\\Windows\\\\\\
  CurrentVersion\\\\\\(Run|RunOnce|Explorer\\\\\\
  (Shell Folders|User Shell Folders)|Policies\\\\\\Explorer\\\\\\Run|
  RunServices|RunServicesOnce)$)|(Microsoft\\\\\\Windows\\\\\\
  CurrentVersion\\\\\\(Run|RunOnce|Explorer\\\\\\
  (Shell Folders|User Shell Folders)|Policies\\\\\\Explorer\\\\\\Run|
  RunServices|RunServicesOnce))"
```

Despite not being set explicitly, this event is also triggered when a registry key is deleted. To filter only events triggered by creation of registry key, the *Operation Type* field is used.

When it comes to the artifacts that are to be retrieved after this alert is triggered, the interesting files are the ones that are set to start by the newly added registry key. In the key value, there can be multiple files present, for example, as arguments to the program set to be launched. Therefore, in the corresponding function within the alert action script, the regular expression is used to fetch all the paths present in the *New Value* field. That is where the newly set value for the registry key is recorded inside the event. It is possible that some fetched files will not be useful for analysis, for example, in the case where *cmd.exe* is the file launched with some arguments, which is also commonly used.

This rule is tested alongside the next two rules.

T1547.001.CMD: Persistence attempt via Registry Run Keys detected in command line

This next rule monitors attempts to gain persistence from the command line. It checks for an occurrence of the strings in the *Process Command Line* field of the event ID *4688*. The rule is similar to the previous one, changing only the event ID, the field in the event and there is no necessity to filter the *Operation Type* field. The regular expression stays the same.

As for the collected artifacts, every file path detected in the command line will be collected as well as the paths present in the *Creator_Process_Name* field.

T1547.001.PSS: Persistence attempt via Registry Run Keys detected in Powershell script

The third implemented rule detects the occurrences of the string in Powershell Script Blocks logged by the event ID *4104*. The rule is also similar to the previous one, changing only the event ID and searching in the *Message* field.

When it comes to artifacts, it is similar to the previous alert as any paths present in the *Message* field are collected.

Testing

To test the detection of the startup folder rule (*T1547.001.STF*), for now it is enough to create any type of file in each of the two folders and see that the alert was triggered and that the corresponding flows were completed successfully. It is also possible to see the flow in the GRR interface and explore the file contents and properties there.

To test the three registry keys alerts, it is enough to use a single script (*testregistrydetection.ps1*). It is a Powershell script that creates a registry entry in all identified registry keys using *cmd.exe* which is invoked by this Powershell script. Therefore, after the script is run, all three alerts are triggered. In addition, a test file is created by the script for each key in a directory with text contents corresponding to the key, which should be visible in the Google Rapid Response interface. The script subsequently deletes these files and registry entries after a 15-second timeout, so that GRR manages to retrieve files.

4.4 Scheduled Task/Job: Scheduled Task (T1053.005)

The detection of abuse of scheduled tasks works on a similar principle as the previous technique concerning the registry. Again, *Windows event logs* are utilized alongside the detection of the command line and Powershell. Still, some additional configurations have to be made and new rules have to be designed.

4.4.1 Additional Endpoint Configuration

The Windows event utilized to log the creation of a scheduled task has theID *4698*. Again, it is a security audit log, and it is not enabled by default. It falls into the category of *Audit Other Object Access Events*, which is found, similarly to the previously set configuration, inside the *Object Access* tab under the *Advanced Audit Policy Configuration*.

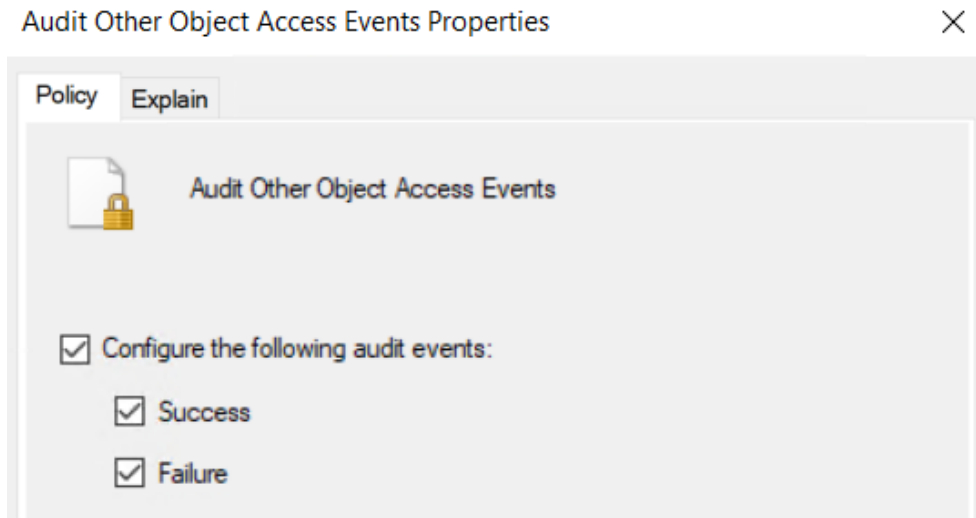


Figure 4.8: Enabling security audit logs of *Other Object Access Events* in order to invoke event ID 4698.

To detect possible malicious activity related to scheduled tasks using the command line and Powershell, the same log events as with the registry keys (4688 and 4104) will be used.

4.4.2 Detection and Alerting

The detection logic of a scheduled tasks activity is similar to the registry. There will be 3 rules implemented. One general alert reacting on the occurrence of the event ID 4698 and two additional alerts checking the command line and Powershell events for a presence of a string indicating manipulation with scheduled tasks.

T1053.005.GEN: A scheduled task was created

This is a very simple alert that is triggered only by the presence of event ID 4698:

```
index=main EventCode=4698
```

As mentioned in 2.2.2.1, the entire XML with the configuration is present in this event, so it is necessary to parse it in the alert action script to fetch some relevant data using GRR, particularly the command scheduled to run. In the XML structure, the command can be found under this series of tags: Task -> Actions -> Exec -> Command.

Alongside the *Command* tag, there can be also an *Arguments* tag, which can contain interesting file locations.

```
<Actions Context="Author">
  <Exec>
    <Command>C:\Windows\System32\notepad.exe</Command>
  </Exec>
</Actions>
```

Figure 4.9: A command located inside an example scheduled task configuration XML logged by the event ID 4698.

Although there is an XML structure in the event log, Splunk does not pass the field as a plain XML string, as there are whitespaces and new line characters added. Therefore, it is easier to just find and extract the string between the *Command* and *Arguments* tags using a regular expression. After that, again, every file path is extracted from this string and acquired using the *FileFinder* GRR flow.

Alongside the command set to run, the configuration file from the `C:\Windows\System32\Tasks` directory will be collected to have it available also in GRR. The location of the config file corresponds to appending the task name to the tasks directory. The task name is parsed as the *Task_Name* field by Splunk.

T1053.005.CMD: Scheduled task activity detected in the command line

The presence of the string *schtasks* is the main detection indicator while monitoring the command line. As mentioned 2.2.2.1, there are also 5 action options possible to utilize while using the *schtasks* utility - */Create*, */Delete*, */Run*, */Query* and */Change*. This alert is designed to detect not only the */Create* flag, but also the */Change* flag as it can be utilized to edit an existing scheduled task, including changing the command that is set to be run:

```
index=main EventCode=4688 Process_Command_Line=*schtasks*
(Process_Command_Line=*/create* OR
Process_Command_Line=*/change*)
```

Again, as an automatic response, any path appearing in the *Process Command Line* field is acquired. There can be cases where the collection is redundant, similarly to the registry command line detection. However, for example, when a concrete configuration file is used as an argument for a scheduled task creation (as in the *AgentTesla* example), it is beneficial to also collect that file. The paths from *Creator_Process_Name* are also collected.

T1053.005.PSS: Scheduled task activity via Powershell detected

This rule utilizes Powershell script block logging and has 2 strings that it scans for in the *Message* field - *New-ScheduledTask* and *Set-ScheduledTask*, which are Powershell cmdlets designated to create and update a scheduled task, respectively. Using this logic, additional cmdlets are also detected (*New-ScheduledTaskAction*, *New-ScheduledTaskPrincipal*, *New-ScheduledTaskSettingsSet*, *New-ScheduledTaskTrigger*), which is fine as they are related to a scheduled tasks activity.

```
index=main EventCode=4104
(Message=*New-ScheduledTask* OR Message=*Set-ScheduledTask*)
```

The automated action once again searches for all paths in the command. Typically, it should fetch the path of the file to be executed and the script itself, as at the end of the message field, the path to the script is logged.

Testing

In this case, the rules will be tested using a command line script (*testsched-cmd.cmd*) and two Powershell scripts. In the command line script, there are two cases covered:

1. A *cmd* command which creates a scheduled task named *TestTasks Task1* for executing notepad.exe every day at 11:00. This command triggers 2 alerts - *T1053.005.GEN* and *T1053.005.CMD*. The notepad file as well as the XML file with the configuration are fetched by GRR.
2. A *cmd* command with the */Change* flag sets the execution file as *calc.exe*. This file triggers only the *T1053.005.CMD* command and the calculator executable is acquired.

After that, the task is deleted.

In the first Powershell script (*testschedpssnew.ps1*), a scheduled task is created using *New-ScheduledTask* to execute Notepad. This should trigger the general alert and a PSS alert, with GRR collecting the script itself, the notepad binary, and the XML configuration file of the task. The second script (*testschedpss_set_delete.ps1*) changes the task using *Set-ScheduledTask* to execute Calculator and then deletes it after a short timeout. The PSS alert is triggered, and GRR should collect the script itself and the calculator binary.

4. IMPLEMENTING DETECTION MECHANISMS

The screenshot displays two File Finder results from GRR. The first result, with ID ADEA924CF49084F9, shows a single file: C:\Windows\System32\calc.exe. The second result, with ID FDD418BD4253088A, shows two files. A table of file details is provided for the second instance:

Path	Hash	Mode	Size	A-time	M-time	C-time	B-time
C:\Windows\System32\notepad.exe	SHA-256	-rwxrwxrwx	196.00 KiB	2024-05-01 15:04:20 UTC	2024-03-07 20:17:28 UTC	2024-03-07 20:17:28 UTC	
C:\Windows\System32\Tasks\TestTasks\Task1	SHA-256	-rw-rw-rw-	3.42 KiB	2024-05-01 15:04:14 UTC	2024-05-01 15:04:14 UTC	2024-05-01 15:04:14 UTC	

Figure 4.10: Example of the data collected by GRR from the command line test case for detection of potential scheduled tasks abuse.

4.5 Windows Management Instrumentation (WMI) Event Subscription (T1546.003)

The used detection mechanisms remain the same as with the previous techniques. The main detection tool will be the native *Windows Event Logs*, supplemented by the command line and Powershell detections.

4.5.1 Additional Endpoint Configuration

As designed, the *Operational* logs from the WMI log source will be utilized. These logs are active by default, the only thing left to set up is to pass it to Splunk. That means editing the *input.conf* file in the Splunk forwarder on the endpoint by adding the following lines:

```
[WinEventLog://Microsoft-Windows-PowerShell/Operational]
disabled = 0
```

After a restart, the logs are visible in Splunk.

4.5.2 Detection and Alerting

T1546.003.GEN: WMI Event Subscription Persistence Attempt Detected

The general alert to detect an attempt to gain persistence using the WMI Event Subscription leverages the event ID *5861* from the WMI operational logs. It alerts when this alert ID is detected along with one of the forms of the subscription namespace strings (*root\subscription*, *root/subscription*) in combination with presence of the name of two commonly abused consumer classes (*ActiveScriptEventConsumer*, *CommandLineEventConsumer*) in the message field:

```
index=main EventCode=5861
(Message = *root/subscription*
 OR Message = *root\subscription*)
(Message = *CommandLineEventConsumer*
 OR Message = *ActiveScriptEventConsumer*)
```

Generally, the whole event filter and event consumer definitions are present in this event log.

The automated action retrieves files from all the Windows paths present in the message field (contains mainly definitions of the filter and the consumer). However, the paths are logged with double backslashes, so, as the first step, all double backslashes are swapped for only single backslash in the automation script. In addition to these files, the whole WMI repository is collected from *C:\System32\wbem\Repository*.

T1546.003.CMD: A suspicious WMI consumer created via command line

This is an additional alert to detect only the creation of a WMI consumer through the command line. Using event ID *4688* and its *Process Command Line* field, it searches for strings similar to those commonly used in previous alert. It only skips the possible creation of a WMI bind to avoid duplicate detections. The consumer is the one monitored because it commonly includes the commands to be executed, including the paths.

```
index=main EventCode=4688 Process_Command_Line = *CREATE*
(Process_Command_Line = *root/subscription*
 OR Process_Command_Line = *root\subscription*)
(Process_Command_Line = *CommandLineEventConsumer*
 OR Process_Command_Line = *ActiveScriptEventConsumer*)
NOT Process_Command_Line = *__FilterToConsumerBinding*
```

Again, all the paths found in the command are automatically acquired into GRR along with the creator process path.

T1546.003.PSS: Suspicious WMI Event Subscription activity launched from Powershell

WMI persistence is often invoked by Powershell. To detect it, script block logging is used, again utilizing the event ID *4104*. As designed, the alert detects presence of one of the cmdlets used to introduce new WMI entries while also checking the commonly present strings as in the previous two alerts:

```
index=main EventCode=4104
(Message = *root/subscription*
 OR Message = *root\\subscription*)
(Message = *New-CmiInstance* OR Message = *Set-WmiInstance*)
(Message = *ActiveScriptEventConsumer*
 OR Message = *CommandLineEventConsumer*)
```

The automatic response again searches for file paths.

T1546.003.MOF: MOF file compiled using mofcomp.exe

The possibility of creating a WMI subscription using a MOF file is detected simply by detecting the presence of *mofcomp* string inside a command logged by event ID *4688*. When an event subscription is created, additional information can be found in the general alert. Otherwise, the potential analyst has to analyze the file MOF file itself from the GRR UI.

```
index=main EventCode=4688 Process_Command_Line = *mofcomp*
```

As it is not guaranteed that there will always be a general alert to pair this one with, the automatic alert action tries to recover the mof file itself. Therefore, it again extracts all the paths present in the command. There is a possibility that the MOF file is passed as an argument in the form of only a relative path, and GRR's capability of finding files is used. With the expression `%%environ_systemdrive%%**15\<file_name>` searching the directories under the system drive recursively up to 15 levels.

Testing

Three test scripts are created to test these four alerts. The first one (*test-wmicmd.cmd*) creates the event filter, the consumer, and the binding by the *wmic* tool, similarly as shown in 2.2.3.1. It is also automatically deleted. The powershell script (*testwmipowershell.ps1*) also adds the three components by using the *Set-WmiInstance* command. It is also automatically deleted after a short timeout. The third script (*testwmihof.cmd*) only launches the *mofcomp.exe* utility with *moftest.mof* file as an argument. The *MOF* file defines the three WMI subscription components.

4.6 BITS Jobs (T1197)

As opposed to the previous techniques, Windows event logs are not very useful to distinguish persistence attempts from other purposes of BITS jobs. Therefore, the detection concentrates on the command line and Powershell.

4.6.1 Additional Endpoint Configuration

Although logs are not used for detection, it is convenient to have them available in log management. As BITS jobs have their own operational logs, it is necessary to update the *input.conf* file on the endpoint to collect the logs:

```
[WinEventLog://Microsoft-Windows-Bits-Client/Operational]
disabled = 0
```

Configurations for other log sources are already prepared from previous techniques.

4.6.2 Detection and Alerting

To detect potential persistence attempts, 2 rules are implemented, one focusing on command line detection and the other one monitoring Powershell script blocks.

T1197.CMD: BITS jobs SetNotifyCmd flag detected

The purpose of this alert is only to detect the presence of the *SetNotifyCmdLine* flag in a command connected to *bitsadmin*. When the flag is present, it indicates an attempt to gain persistence. Again, the event ID 4688 is utilized.

```
index=main EventCode=4688 New_Process_Name="*bitsadmin*"
Process_Command_Line="*SetNotifyCmdLine"
```

Regarding artifacts, all the paths present in the command and creator process are acquired, however, without the BITS state files which are present inside of the *C:\ProgramData\Microsoft\Network\Downloader* directory as the GRR client does not have permissions to collect it.

T1197.PSS: BITS job with NotifyCmdLine flag detected through Powershell

To detect possible persistence originating from Powershell commands, a similar principle as in the previous command is used. Script block logging is used to detect whether either *Start-BitsTransfer* or *Set-BitsTransfer* command was used with the flag *NotifyCmdLine* which indicates possible persistence attempt. The *Set-BitsTransfer* command is detected because it can be utilized to add the notify command to an existing job.

```
index=main EventCode=4104
(Message="*Start-BitsTransfer*" OR Message="*Set-BitsTransfer*")
Message="*NotifyCmdLine"
```

Again, all the files from the Message field are acquired.

Testing

Two scripts are created that simulate persistence attempts through bits jobs. *testbitscmd.cmd* uses the command line based on the method described in 2.2.4.1 and *testbitsps1.ps1* utilizes only the *Start-BitsTransfer* cmdlet to set up the persistent job. Both scripts at first create a file to tie with the job (it can be any file), create the BITS job and then, after timeout, complete the job and delete the file to return the state before the testing.

4.7 Windows Service (T1543.003)

For monitoring this technique, it is necessary to add some additional configurations to obtain security logs, which are used to create alerts together with the command line and Powershell logs similarly as in previous techniques.

4.7.1 Additional Endpoint Configuration

The only new log source that is used for detection is the Windows event log ID 4697. It is a security log that is not enabled by default and does not fall into any category that was manually enabled earlier. It belongs to the category of *Audit Security System Extension*, which falls under the *System* tab inside the *Advanced Audit Policy Configuration*.

4.7.2 Detection and Alerting

There are three alerts introduced following a scheme utilized in some previous techniques with one general alert utilizing a dedicated Windows security event log and two additional alerts monitoring commands, cmdlets, and their parameters.

T1543.003.GEN: New service was created on the system

This alert reacts on every newly created service on the machine.

```
index=main EventCode=4697
```

As of collecting interesting files, it collects the binary scheduled to run from the *Service File Name* field. In case there is a registry subkey created inside the *Services* key, a RegistryFinder flow is invoked which tries to collect an entry with the name of the service obtained from the *Service Name* field.

4.7. Windows Service (T1543.003)

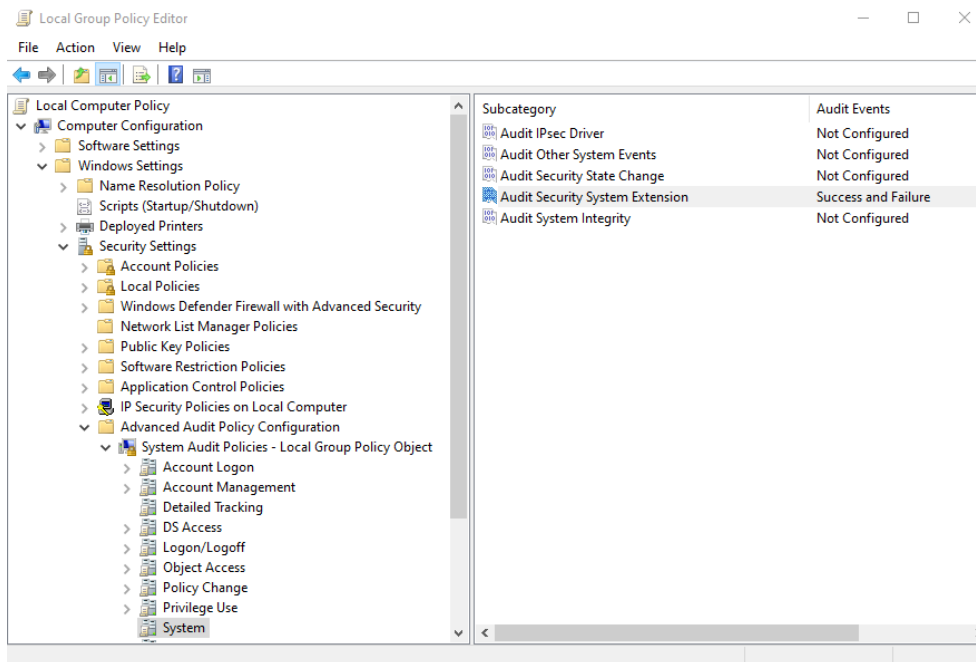


Figure 4.11: Enabling the Windows event ID 4697 via the group policy editor.

T1543.003.CMD: Windows service creation or modification detected from command line

Creation of a new service can also be monitored via command line by searching for the `sc` command. The alert is triggered when it is run with the flags `create` or `config` and the `binpath` argument.

```
index=main EventCode=4688
(Process_Command_Line="*sc *"
 OR Process_Command_Line=*sc.exe*)
(Process_Command_Line=*create*
 OR Process_Command_Line=*config*)
(Process_Command_Line=*binpath*)
```

Again, all possible paths are collected from the command line and the creator process fields.

T1543.003.PSS: New service created through Powershell

Powershell script blocks are monitored for usage of the `New-Service` cmdlet along with the `BinaryPathName` parameter.

```
index=main EventCode=4104 Message=*New-Service*
Message=*BinaryPathName*
```

As with previous alerts monitoring powershell, all paths found in the *Message* field are collected.

Testing

Two test scripts are introduced, one (*testservicescmd.cmd*) using the *sc* command to create a service which runs the Windows calculator and the other (*testservicesps1.ps1*) using *New-Service* to run Notepad.

4.7.3 Observations

In a case where the persistence attempt is conducted via command line or Powershell, two alerts (and if Powershell invokes the command line, even three alerts) are triggered for a single action. The downside may be that the information about only one persistence attempt is not together. However, as mentioned before, the basic Splunk version does not offer sophisticated alert management by default, so in this case, it would be the responsibility of the analyst to notice two related alerts.

When it comes to the automated artifact acquisition, utilization of separate events is beneficial, as it can collect common artifacts anytime the general persistence attempt via registry alert is triggered. Besides, it allows to acquire additional material depending on whether the attempt was triggered either via command line or Powershell. There is of course room for implementing detection of other methods.

It can occur that a GRR flow to fetch the same file will be triggered when multiple alerts hit for the same action on the endpoint. However, each rule can be triggered separately and this redundance is not a problem in GRR. Yes, there will be multiple redundant flows visible, however, while observing the virtual filesystem of the endpoint which contains only acquired files, it will not cause any confusion for the analyst as only one copy of the file is visible there. Still, it is desirable to have more artifacts ready for the analyst to manually filter out, rather than miss an important detail.

Additional Testing and Observations

After completing the practical solution, it is interesting to observe how it behaves during a real scenario. The objective is to run a few real-world malware samples and observe how the solution works. The samples are downloaded from *MalwareBazaar*. After running each sample, the Azure machine is restored to the clean state as before. Before running the samples, it is necessary to disable Microsoft Defender on the endpoint. In addition, it is also convenient to see if some false positives are spotted for eventual whitelisting. Lastly, it is worth to summarize some observations about the work in general.

5.1 TrickBot

The first sample selected is a random sample of TrickBot malware with this SHA1 hash: `34511192ce0d8d8bc7fdb21eff15d8475f53a765`. It was distributed in a phishing campaign posing as an invoice while in reality being an executable file. [126]

After running the file, it appears that nothing happens to the user, however, in the background, an interesting activity is observed. The detection triggered *six* alerts through two minutes after the malware was run.

Time	Alert name	App	Type	Severity
2024-05-08 08:52:38 UTC	T1053.005.CMD: Scheduled task activity detected in the command line	search	Real-time	High
2024-05-08 08:52:19 UTC	T1053.005.CMD: Scheduled task activity detected in the command line	search	Real-time	High
2024-05-08 08:51:47 UTC	T1053.005.CMD: Scheduled task activity detected in the command line	search	Real-time	High
2024-05-08 08:51:28 UTC	T1547.001.GEN: Persistence attempt via Registry Run Keys	search	Real-time	Medium
2024-05-08 08:51:28 UTC	T1053.005.GEN: A scheduled task was created	search	Real-time	Medium
2024-05-08 08:51:28 UTC	T1053.005.CMD: Scheduled task activity detected in the command line	search	Real-time	High

Figure 5.1: Alerts triggered after running a sample of the TrickBot malware.

5. ADDITIONAL TESTING AND OBSERVATIONS

We can observe that there are *four* separate alerts for *T1053.005.CMD*, indicating that the malware is abusing attempting to abuse scheduled tasks through the command line. At the time of the first command line alert, there is also the general alert *T1053.005.GEN* visible, indicating the creation of a scheduled task. Together with the previous two alerts, the malware also triggers the *T1547.001.GEN* alert which detects an attempt to gain persistence via the registry run keys. After that, in the course of around a minute, three more *T1053.005.CMD* alerts are triggered.

It is interesting to look at the command present in the event log ID *4688*, which originated the first alert:

```
"C:\Windows\System32\schtasks.exe" /Create /TN
"Updates\HRggTwDJmFoFS" /XML
C:\Users\endpointuser1\AppData\Local\Temp\tmp8515.tmp"
```

There were some artifacts acquired through GRR. Unfortunately, although the flow attempted to fetch the *tmp8515.tmp* file, it was probably deleted sooner than the GRR agent managed to collect it. While the legitimate *schtasks.exe* binary was obtained a few times, which is an expected behaviour, the launched sample file itself was collected along with a malicious binary *C:\Program Files (x86)\vkl.exe* which was created by the malware. This file was set to be run from the registry run key and was even the originator of the newer *T1053.005.CMD* alerts. These are very useful files to have available right away for the responding analyst. There are also a lot of information available from the logs alone.

5.2 AgentTesla

A malware sample signed as *AgentTesla* on *MalwareBazaar* was also tested. This sample posed as a PDF file, however it was an executable. The SHA1 hash is *fe25931e12b1f5807b95cf222cd9ee74c2cb7ea2*. [127]

Again, after running the file, nothing appeared, however, *four* alerts were triggered.

Time	Alert name	App	Type	Severity
2024-05-08 10:18:04 UTC	T1547.001.GEN: Persistence attempt via Registry Run Keys	search	Real-time	Medium
2024-05-08 10:17:54 UTC	T1547.001.GEN: Persistence attempt via Registry Run Keys	search	Real-time	Medium
2024-05-08 10:17:54 UTC	T1053.005.GEN: A scheduled task was created	search	Real-time	Medium
2024-05-08 10:17:54 UTC	T1053.005.CMD: Scheduled task activity detected in the command line	search	Real-time	High

Figure 5.2: Alerts triggered after running a sample of the AgentTesla malware.

Unsurprisingly, the malware attempted to gain persistence through both, registry run keys and scheduled tasks. At one moment, there are three alerts

triggered, *T1547.001.GEN*, *T1053.005.GEN* and *T1053.005.CMD*. After approximately 10 seconds, another manipulation with registry run keys was detected. The sample was left running for around 5 minutes and no other alerts were observed.

The initial command for gaining persistence through scheduled tasks is very interesting:

```
"C:\Windows\System32\cmd.exe" /c schtasks /create /f /sc onlogon
/rl highest /tn "svchost" /tr
' "C:\Users\endpointuser1\AppData\Roaming\svchost.exe"' & exit
```

The malware probably created a file posing as *svchost.exe*, however, in the *AppData* directory of the user. The name of the scheduled task is also *svchost*. This is a nice attempt to hide the malicious task by constructing the name in a way to hide between legitimate tasks and processes. There was also a registry run key created for this binary. The last alert was generated to create a registry run key for a new binary created by the malware located in *C:\Users\endpointuser1\AppData\Roaming\veplorers\veplorers.exe*.

File Finder admin – 2024-05-08 10:17:56 UTC
4E1787A30DFBCD21

Flow arguments

C:\Windows\System32\cmd.exe + 2 more




Path	Hash	Mode	Size
 C:/Windows/System32/cmd.exe	SHA-256	-rwxrwxrwx	283.00 KiB
 C:/Users/endpointuser1/AppData/Roaming/svchost.exe	SHA-256	-rwxrwxrwx	673.91 KiB
 C:/Users/endpointuser1/Downloads/sample mal/df9e900bc2aba3462d0b9d2fb4e81719 604f4c63871a2225edce136c140e8fc8.exe	SHA-256	-rwxrwxrwx	673.91 KiB

Figure 5.3: Sample of one of the flows collecting artifacts from the test run of the AgentTesla malware.

All the interesting files were collected automatically by GRR, including the malware sample itself, the fake *svchosts.exe* file and the newly created

vexplorers.exe binary. Again, as expected, the legitimate binary *cmd.exe* was collected by two flows as a side effect.

5.3 RedLineStealer

The next sample chosen was a random binary with the *RedLineStealer* tag and SHA1 hash *b150d9e85c5991d2e6d9699edf37a37fc3027eca*. [128].

This sample did not trigger any alerts in the first 5 minutes of its runtime. After examining the same sample in the *Any.Run* sandbox, it also did not show any technique related to persistence, so chances are that this particular sample does not attempt any of the techniques implemented in the detection.

5.4 Remcos

Another recently active malware is *Remcos*. A random sample posing as a PDF file was again selected and downloaded from *MalwareBazaar*. The SHA1 hash is *4258ca2ef7d11d6dc1f56127118685e838f84085*. [129]

There were only 2 alerts activated after running the sample.

Time	Alert name	App	Type	Severity
2024-05-08 14:47:00 UTC	T1053.005.GEN: A scheduled task was created	search	Real-time	Medium
2024-05-08 14:47:00 UTC	T1053.005.CMD: Scheduled task activity detected in the command line	search	Real-time	High

Figure 5.4: Alerts triggered after running a sample of the Remcos malware.

Apparently, this *Remcos* sample uses only a simple persistence attempt through scheduled tasks using a command. The two alerts were triggered at the same time, one of them being *T1053.005.CMD* and the second one being *T1053.005.GEN*. The command used for establishing persistence is exactly as expected from the analysis:

```
"C:\Windows\System32\schtasks.exe" /Create  
/TN "Updates\iqdSDNHzekt"  
/XML "C:\Users\endpointuser1\AppData\Local\Temp\tmp1486.tmp"
```

The sample file was collected within the alert action reacting on the command line alert. The *tmp1486.tmp* file itself was not acquired as, again, it was probably deleted too quickly for GRR to obtain it. However, the configuration was logged in the event ID *4698*, which was the subject of the triggered general alert. From there, GRR was able to fetch the file *iqdSDNHzekt.exe* dropped into the *AppData* folder of the user that was set to run by the scheduled task.

5.5 QBot

A random *QBot* sample was tested on the endpoint. It had the following SHA1 hash: *2bf60b4709e1e653ad5427761ba70c7b6c22b8ba*. [130] There were no alerts triggered. After checking the logs and some public sandboxes, it is possible that there was not any attempt for gaining persistence carried out. If yes, it went undetected even by the sandboxes.

5.6 Detecting Possible False Positives

In a real environment where the endpoint is used for example by an employee or member of an organization, there are some tools and applications the company uses on a daily basis. These are legitimate applications that often need to perform certain activities that can be detected by an alert targeted to discover malware. Hypoteticaly, there can be hundreds of machines being monitored and performing these legitimate tasks. That could result in many false positive alerts, which would be overwhelming for the analyst. Effective whitelisting can therefore filter out many unnecessary work.

The goal is to observe not only alerts triggered from the behavior of a clean Windows machine, it is also beneficial to observe alerts triggered by some commonly used applications. For demonstration, *Google Chrome* and *Microsoft Office* were installed on the clean endpoint machine and triggered alerts were observed both during the installation and during the startup of the machine. During the installation, there were around 20 alerts triggered. The alerts belonged to one of these categories: *T1053.005.CMD*, *T1053.005.GEN* and *T1543.003.GEN*. As an example, entries in the *Service File Name* field inside the event ID *4697* detected while installing Google Chrome are shown in the table 5.1.

All whitelisted entries can be observed in the attached files inside the *whitelists* directory. There are whitelists for 4 alerts:

- *T1053_005_CMD_whitelist.csv* - whitelisting based on the *Process Command Line* and *Creator Process Name* fields of the event ID *4688*
- *T1053_005_GEN_whitelist.csv* - whitelisting based on the *Command* tag of the *Message* field of the event ID *4698*
- *T1543_003_GEN_whitelist.csv* - whitelisting based on the *Service File Name* field of the event ID *4697*
- *T1547_001_GEN_whitelist.csv* - whitelisting based on the *Object Name*, *Process Name* and *Object Value Name* fields of the event ID *4657*

The whitelists have to be implemented into queries in the saved searches and that is done via Splunk lookups. It is necessary to place a *csv* file into the

5. ADDITIONAL TESTING AND OBSERVATIONS

Service File Name
C:\Program Files (x86)\Google\Update\GoogleUpdate.exe
C:\Program Files (x86)\Google\Update\GoogleUpdate.exe /medsvc
C:\Program Files\Google\Chrome\Application\124.0.6367.156\elevationservice.exe
"C:\Program Files (x86)\Google\GoogleUpdater\126.0.6462.0\updater.exe" -system -windows-service -service=update-internal
"C:\Program Files (x86)\Google\GoogleUpdater\126.0.6462.0\updater.exe" -system -windows-service -service=update
"C:\Program Files\Common Files\Microsoft Shared\ClickToRun\OfficeClickToRun.exe" /service
"C:\Program Files (x86)\Microsoft OneDrive\23.038.0219.0001\OneDriveUpdaterService.exe"
"C:\Program Files (x86)\Microsoft OneDrive\23.038.0219.0001\FileSyncHelper.exe"

Table 5.1: Service File Names observed through the installation of Google Chrome

.../search/lookups directory on the server. Then, in the query, the following search string is appended after the already existing alert query (the alert logic is not changed in any way):

```
NOT [<whitelist>.csv | fields <fields_to_match>]
```

When whitelisting the *T1053.005.GEN* alert, it is also necessary to parse the *Command* tag from the message field. That is done using the *rex* Splunk command and the result then looks as follows:

```
index=main EventCode=4698 | rex field=Message
"<Command>(?(Extracted_Command).*?)</Command>"
| search NOT [|inputlookup T1053_005_GEN_whitelist.csv
| fields Extracted_Command]
```

There are cases when double quotes need to be matched literally. Splunk is not able to do that even when escaped in the lookup file. Therefore, a workaround has to be used, which means adding a single quote to the beginning of the line and adding a *rex* in *sed* mode into the Splunk query as follows:

```
NOT [|inputlookup <whitelist>.csv | fields <field>
| rex field=<field> mode=sed "s/^\`\"/\"/"]
```


5.7 Final Observations

After finalizing both the theoretical preparation and the detection mechanisms itself, there are some specific observations, problems, and generally things worth noting.

Firstly, when it comes to the theoretical foundation and resources, the *MITRE ATT&CK* matrix proves itself to be a very useful resource as far as classification concerns. It can provide some general information about the usage and working mechanisms along with listing the threat actors that have used the particular technique. However, to dive into a greater detail, one has to rely on additional sources, often being articles and analyses performed by private companies. To mention one specifically, *RedCanary* was very useful for researching different types of information, including various statistics, analyses of malware techniques or thorough analyses of one specific malware sample.

Regarding the topic of researching the persistence techniques themselves, the fact concluded from Section 2.1 could be observed very easily. The two most popular techniques are without a major doubt the two following: *Scheduled Task/Job: Scheduled Task (T1053.005)* and *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)*. During the theoretical research of the techniques, there were many different sources describing these two techniques in a great detail as well as many examples and analyses of many malware samples. Researching detailed information about the other techniques was significantly more difficult. What is more, there are not many write-ups on real-world malware samples that utilize these techniques. That also showed during the final testing in this chapter. The malware samples tested belong to popular and recently active malware families and all of them employed one of these two techniques, or even both. Attempts to find samples abusing the other techniques to run them through the created detection mechanism were also without success.

When it comes to the practical implementation, the *Azure* environment was convenient for this type of laboratory. It allowed all the necessary features, one of the main ones being taking snapshots and restoring the virtual machines after some mistakes in their configuration or after running a malware sample. The usage of *Azure* did not bring any limitations.

The log management solution, Splunk, is suitable for the environment of this extent. To practically manage alerts in a larger real environment, it would be certainly more comfortable to use the *Enterprise Security* extension, or to employ a SOAR solution. Regarding the implementation itself, deploying Splunk was really straightforward. The most complicated part was to implement the custom alert action, which had many very specific requirements for contents of files, their names and various other conditions that had to be met. Some of them were not very well documented. Moreover, it can be a problem to use additional Python packages for an alert action. It was necessary to implement a workaround. It is also worth noting that the capability of real-

time alerting is limited in Splunk as it takes up a lot of resources. However, it was enough for the extent of the implemented environment. It would not be a major problem to change the alerts and their respective actions from real-time to scheduled.

Moving on to the endpoint configurations, there were mostly no problems. The choice of the source of logs for the detection mechanism came to the native *Windows Event Logs*. Most of the time, Microsoft's documentation was sufficient to discover what configurations are necessary to make to set up appropriate log flow. However, it would be interesting to see how, for example, usage of Sysmon would change the configuration difficulty. As of information provided by the *Windows Event Logs*, everything necessary for was present there.

Google Rapid Response was implemented to automatically collect interesting files and various artifacts from the endpoint. The biggest setback of the whole work came when trying to set up communication between the GRR client and Server. As it turned out, the problem was in the values of IP addresses in the configuration of both the server and the client. The installation prompt was a bit misleading and the information about whether the configuration expects local or public IP address was not clear. Documentation and other public sources showed only usage in a local environment. At the end, the solution was a combination of both types of addresses with the necessity to modify part of the configuration directly on the endpoint. It is possible that these issues were connected with operating in *Azure* but that is only a speculation. Resolving this issue required a very long time.

Lastly, regarding the GRR functionality itself, it is a very useful tool and it has many pre-built configurations of various artifacts that can be acquired in just seconds. However, there were some imperfections that had an impact on this work. A very good thing is that an API exists. However, that has almost no documentation and aside from very basic examples. all the features had to be read from *protobuf*⁶ definitions. Another feature that would be very useful for this work would be the ability to name or at least tag the generated flow to better pair it with an alert.

5.8 Future Work

The theoretical foundation as well as the practical solution meets the goals that were set and works well to serve the defined purpose. It also leaves room for future exploration and expansion. Although the most commonly occurring persistence techniques were analyzed, there is still room to add more and less known techniques while also implementing them in the detection mechanism. That is also true for techniques belonging to the other tactics in the *MITRE*

⁶Protocol Buffers are language-neutral, platform-neutral extensible mechanisms for serializing structured data. [131]

ATT&CK matrix that were out of the scope of this work. By doing so, a robust solution capable of detecting various malware samples and classifying them based on the *MITRE ATT&CK* techniques would be created.

There is a possibility of scaling also in the environment itself, more endpoints can be added and be monitored. The designed rules and acquisition logic can be used in the same way in a real, larger, environment, only adjusting the infrastructure accordingly to the size of the environment.

The topic of automatic response and artifact acquisition was observed to be very interesting and useful, and it is certainly worth exploring into a greater detail. Various tools could be tested and evaluated. Regarding GRR, there are also some interesting features that were not explored in this work as they did not fit into the solution. Another interesting focus area could be exploring automatic acquisition capabilities from the perspective of forensic analysis as distinct from the perspective of an analyst performing basic triage and alert analysis.

Lastly, it would be interesting to use this solution against a larger set of real-world malware samples to discover what persistence techniques they use and what activity is detected.

Conclusion

The goals of this work were to research various persistence techniques and analyze them in a greater detail, utilize the information to implement a mechanism to detect malware samples employing these techniques while exploring the possibilities of an automatic artifact collection.

The goals were fulfilled, a research concluding with the most commonly used malware persistence techniques was conducted with some techniques later analyzed in detail, containing their working mechanisms, detection opportunities, relevant artifacts and some real-world examples.

The above mentioned information were then used to build the detection solution. To achieve that, at first it was necessary to create a laboratory environment in the cloud and deploy suitable tools. This did not go without problems, there were many challenges that needed to be tackled while making everything work.

The work also provides a practical output, including all configuration files and scripts made specifically for the created environment, being the endpoint machine, the log management tool and the tool for the automated artifact acquisition. An important output is a set of rules detecting various persistence techniques along with whitelists resulting from the behaviour of some common legitimate applications. Scripts used for automated acquisition of artifacts are also a part of the solution. Additional scripts for testing of the detection mechanisms were also created.

In the last chapter, some real-world malware samples were run on the endpoint machine and detected by the detection solution. Lastly, some general observations were described in detail along with opportunities for future expansion of the covered topics.

Bibliography

- [1] *The future of ransomware: Inside Cisco Talos threat hunters*. Cisco. Online. [no date]. Available from: <https://www.cisco.com/site/us/en/learn/topics/security/what-is-malware.html> [Accessed 6 December 2024].
- [2] Man Ho Au; Kim-Kwang Raymond Choo. *Mobile Security and Privacy*. Boston: Syngress, 2017. 978-0-12-804629-6.
- [3] NOVICK, Ari. *Persistence Techniques That Persist*. Cyberark. Online. 3 February 2023. Available from: <https://www.cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist> [Accessed 6 December 2024].
- [4] *Persistence, Tactic TA0003 - Enterprise — MITRE ATT&CK®*. MITRE. Online. 2024. Available from: <https://attack.mitre.org/tactics/TA0003/> [Accessed 6 December 2023]
- [5] DAULAGUPHU, Satyajit. *11 Critical Malware Persistence Mechanisms You Must Know*. Online. 2022. Available from <https://tech-zealots.com/malware-analysis/malware-persistence-mechanisms/> [Accessed 6 December 2023]
- [6] *MITRE ATT&CK®*. MITRE. Online. 2024. Available from: <https://attack.mitre.org/> [Accessed 6 December 2023]
- [7] *What is the MITRE ATT&CK Framework?*. Trellix. Online. 2024. Available from: <https://www.trellix.com/security-awareness/cybersecurity/what-is-mitre-attack-framework/> [Accessed 6 December 2023]
- [8] Villalón-Huerta, Antonio; Marco-Gisbert, Hector; Ripoll-Ripoll, Ismael. *A Taxonomy for Threat Actors' Persistence Techniques*. *Computers & Security*, Volume 121, 2022, 102855, ISSN 0167-4048. Available from: <https://>

- www.sciencedirect.com/science/article/pii/S0167404822002498 [Accessed 12 December 2023]
- [9] GITTINS, Zane & SOLTYS, Michael. *Malware Persistence Mechanisms*. Procedia Computer Science: 176. 88-97. 10.1016/j.procs.2020.08.010.
- [10] VAN NIELEN, Jorik Jaromir. *Dynamic Detection and Classification of Persistence Techniques in Windows Malware*. University of Twente. Online. 22 May 2023. Available from: https://essay.utwente.nl/94945/1/van%20Nielen_MA_EEMCS.pdf
- [11] WEBB, M. S. *Evaluating Tool Based Automated Malware Analysis Through Persistence Mechanism Detection*. Kansas State University. Online. 2018. Available from: <https://krex.k-state.edu/bitstream/handle/2097/38783/MatthewWebb2018.pdf?sequence=3&isAllowed=y>
- [12] *What is the MITRE ATT&CK Framework?*. Blackberry. Online. 2024. Available from: <https://www.blackberry.com/us/en/solutions/endpoint-security/mitre-attack> [Accessed 12 December 2023]
- [13] YUCEEL, Huseyin Can. *Scheduled Task/Job - the most used MITRE ATT&CK persistence technique*. Online. 24 May 2023. Available from: <https://www.picussecurity.com/resource/scheduled-task/job-the-most-used-mitre-attck-persistence-technique> [Accessed 15 December 2023]
- [14] *The Red Report 2023*. Picus Security. Online. 2023. Available from: <https://www.picussecurity.com/hubfs/Red%20Report%202023/RedReport2023-Picus.pdf?hsCtaTracking=a6b9f00e-0309-4be1-9d07-ec3bbd7ffe4c%7Ce5400384-e2ec-4a46-97ac-bbdfedf8c6e2> [Accessed 15 December 2023]
- [15] FRENCH, David. *Adversary tradecraft 101: Hunting for persistence using Elastic Security (Part 1) — Elastic Security Labs*. Online. 1 June 2022. Available from: <https://www.elastic.co/security-labs/hunting-for-persistence-using-elastic-security-part-1> [Accessed 15 December 2023]
- [16] DAULAGUPHU, Satyajit. *11 Critical Malware Persistence Mechanisms You Must Know*. Online. 2022. Available from <https://techzealots.com/malware-analysis/malware-persistence-mechanisms/> [Accessed 15 December 2023]
- [17] *Common malware persistence mechanisms — Infosec*. Online. 13 June 2016. Available from: <https://resources.infosecinstitute.com/topics/malware-analysis/common-malware-persistence-mechanisms/> [Accessed 15 December 2023]

-
- [18] MACRORAMILI. *Malware Persistence Locations*. Online. 23 September 2023. Available From: <https://marcoramilli.com/2023/09/23/malware-persistence-locations-windows-and-linux/> [Accessed 15 December 2023]
- [19] LE, Tran Duc & DINH, Duy & NGUYEN T. H., Phuoc & MUTHANNA, Ammar & ABD EL-LATIF, Ahmed. *Exploring Common Malware Persistence Techniques on Windows Operating Systems (OS) for Enhanced Cybersecurity Management*. 2023. Cybersecurity Management in Education Technologies: 10.1201/9781003369042-7. Available from: https://www.researchgate.net/publication/375531872_Exploring_Common_Malware_Persistence_Techniques_on_Windows_Operating_Systems_OS_for_Enhanced_Cybersecurity_Management
- [20] ANANIN, Vlad. *Malware Trends Report: Q4, 2023*. ANY.RUN's Cybersecurity Blog. Online. 27 December 2023. Available from: <https://any.run/cybersecurity-blog/malware-trends-q4-2023/> [Accessed 7 January 2024]
- [21] *In-the-Wild*. Trendmicro. [no date]. Online. Available from: <https://trendmicro.com/vinfo/us/security/definition/in-the-wild> [Accessed 9 January 2024]
- [22] *Avast Q4/2023 Threat Report*. Avast Threat Labs. Online. 7 February 2024. Available from: <https://decoded.avast.io/threatresearch/avast-q4-2023-threat-report/> [Accessed 9 January 2024]
- [23] *MalwareBazaar - Malware sample exchange*. Online. [no date]. Available from: <https://bazaar.abuse.ch/> [Accessed 12 January 2024]
- [24] *MalwareBazaar - Statistics*. Online. [no date]. Available from: <https://bazaar.abuse.ch/statistics>. [Accessed 12 January 2024]
- [25] MORE, Ghanshyam. *Catching the RAT called Agent Tesla — Qualys Security Blog*. Online. 23 December 2022. <https://blog.qualys.com/vulnerabilities-threat-research/2022/02/02/catching-the-rat-called-agent-tesla> [Accessed 13 January 2024]
- [26] SPLUNK. *Inside the mind of a 'Rat' - Agent Tesla Detection and Analysis — Splunk*. Online. 16 November 2022. Available from: https://www.splunk.com/en_us/blog/security/inside-the-mind-of-a-rat-agent-tesla-detection-and-analysis.html [Accessed 13 January 2024]
- [27] *Agent Tesla, Software S0331 — MITRE ATT&CK®*. MITRE. Online. 2024. Available from: <https://attack.mitre.org/software/S0331/> [Accessed 13 January 2024]

- [28] *RedLine Stealer Malware detailed analysis 2024*. Gridinsoft LLC. Online. 15 February 2024. Available from: <https://gridinsoft.com/spyware/redline> <https://gridinsoft.com/spyware/redline> [Accessed 20 January 2024]
- [29] *Redline Stealer*. Viettel Security. Online. February 2022. Available from: <https://viettelcybersecurity.com/wp-content/uploads/2022/02/Report-Redline-Stealer.pdf> [Accessed 20 January 2024]
- [30] HOWARD, Tricia. *RedLine is on track, Next stop - Your credentials*. Cynet. Online. 15 January 2024. Available from: <https://www.cynet.com/attack-techniques-hands-on/redline-is-on-track-next-stop-your-credentials/> [Accessed 20 January 2024]
- [31] PLAMBECH, Rasmus. *Emerging threat: RedLine Stealer malware outbreak - a comprehensive guide to anatomy, detection, and response*. Logpoint. Online. 30 October 2023. Available from: <https://www.logpoint.com/en/blog/redline-stealer-malware-outbreak/> [Accessed 20 January 2024]
- [32] GAGANDEEP, Mehta. *Redline Infostealer Analysis (Part 1)*. Threat-Spike blog. Online. 20 January 2023. Available from: <https://www.threatspike.com/blogs/redline-part-1> [Accessed 20 January 2024]
- [33] MEIR, Matan. *FormBook - Yet another stealer malware*. SentinelOne. Online. 27 October 2022. Available from: <https://www.sentinelone.com/blog/formbook-yet-another-stealer-malware/> [Accessed 25 January 2024]
- [34] PALAZOLO, Gustavo. *New Formbook campaign delivered through phishing emails*. Netskope. Online. 11 March 2022. Available from: <https://www.netskope.com/blog/new-formbook-campaign-delivered-through-phishing-emails> [Accessed 25 January 2024]
- [35] CYSIV Inc. *Threat Report: Formbook Infostealer*. Online. 23 February 2021. Available from: <https://www.forescout.com/resources/formbook-infostealer/> [Accessed 25 January 2024]
- [36] ZHANG, Xiaopeng. *Deep analysis: FormBook new variant delivered in phishing campaign - Part III - FortiGuard Labs*. Fortinet Blog. Online. 27 April 2021. Available from: <https://www.fortinet.com/blog/threat-research/deep-analysis-formbook-new-variant-delivered-in-phishing-campaign-part-iii> [Accessed 25 January 2024]
- [37] COHEN, Hido. *Remcos Trojan: Analyzing the Attack Chain*. Morphisec. Online. 3 July 2023. Available from: <https://blog.morphisec.com/remcos-trojan-analyzing-attack-chain> [Accessed 2 February 2024]

-
- [38] MCAFEE. *Peeling back the layers of RemcosRat malware*. McAfee Blog. Online. 9 December 2023. Available from: <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/peeling-back-the-layers-of-remcosrat-malware/> [Accessed 2 February 2024]
- [39] CYFIRMA. *The persistent danger of REMCOS RAT - CYFIRMA*. CYFIRMA. Online. 23 August 2023. Available from: <https://www.cyfirma.com/outofband/the-persistent-danger-of-remcos-rat/> [Accessed 2 February 2024]
- [40] *Remcos Malware Information*. TrendMicro. Online. 30 December 2019. Available from: https://success.trendmicro.com/dcx/s/solution/1123281-remcos-malware-information?language=en_US&sfdcIFrameOrigin=null [Accessed 2 February 2024]
- [41] *9c4031e2da81b9dc84b39c96055e5e628f4f4fa8db1ab0e26b1756a1cc1af936 - Triage*. TRIA.GE. Online. Available from: <https://tria.ge/240220-vp9nxace72>. [Accessed 10 February 2024].
- [42] *Windows Management Instrumentation, Technique T1047 - Enterprise - MITRE ATT&CK®*. MITRE. Online. 2024. Available from: <https://attack.mitre.org/techniques/T1047/> [Accessed 12 February 2024].
- [43] *Hijack Execution Flow: DLL Side-Loading, Sub-Technique T1574.002 - Enterprise - MITRE ATT&CK®*. MITRE. Online. 2024. Available from: <https://attack.mitre.org/techniques/T1574/002/> [Accessed 15 February 2024]
- [44] *Security 101: How Fileless attacks work and persist in systems*. TrendMicro. Online. 30 April 2020. Available from: trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/security-101-how-fileless-attacks-work-and-persist-in-systems [Accessed 15 February 2024]
- [45] BAKER, Kurt. *What is Fileless Malware? - CrowdStrike*. CrowdStrike. Online. 17 April 2023. Available from: <https://www.crowdstrike.com/cybersecurity-101/malware/fileless-malware/> [Accessed 15 February 2024]
- [46] Business Bliss Consultants FZE. *The Rise of Fileless Malware and Attack Techniques*. ukdiss.com. Online. November 2018. Available from: <https://ukdiss.com/examples/fileless-malware-attack-techniques.php> [Accessed 15 February 2024]
- [47] MINER, Madison. *What is an Artifact in Cyber Security? - Systems Solution, Inc. (SSI)*. Online. 2 March 2021. Available from: insider.ssinet.com/insights/what-is-an-artifact-in-cyber-security [Accessed 24 February 2024]

BIBLIOGRAPHY

- [48] *Windows Event Logs*. Forensafe. Online. 28 June 2022. Available from: https://forensafe.com/blogs/event_logs.html [Accessed 24 February 2024]
- [49] *What is a Windows Event Log? - IT Glossary — SolarWinds*. Solarwinds Blog. Online. [no date]. <https://www.solarwinds.com/resources/it-glossary/windows-event-log> [Accessed 24 February 2024]
- [50] YASAR, Kinza and GILLIS, Alexander S. *Windows event log. Search-WindowsServer*. Techtarget. Online. 14 March 2023. Available from: [techtarget.com/search/windowsserver/definition/Windows-event-log](https://www.techtarget.com/search/windowsserver/definition/Windows-event-log) [Accessed 24 February 2024]
- [51] *Collecting logs from Windows Event Log — NXLog Docs*. NXLog Docs. Online. 2 March 2023. Available from: <https://docs.nxlog.co/integrate/windows-eventlog.html> [Accessed 24 February 2024]
- [52] CHARTER, Brandon. *EVTX and Windows Event Logging*. GIAC. Online. 2008. <https://www.giac.org/paper/gcia/2999/evtx-windows-event-logging/115806> [Accessed 24 February 2024]
- [53] FISHER, Tim. *What is the Windows Registry?* Lifewire. Online. 13 June 2023. Available from: <https://www.lifewire.com/windows-registry-2625992> [Accessed 27 February 2024]
- [54] FREDA, Anthony. *What is the Windows Registry and how does it work?* Avast. Online. 23 February 2023. Available from: [avast.com/c-windows-registry](https://www.avast.com/c-windows-registry) [Accessed 27 February 2024]
- [55] WHIMS, Steve et al. *Registry Hives - Win32 apps — Microsoft Learn*. Online. 7 January 2021. Available from: <https://learn.microsoft.com/cs-cz/windows/win32/sysinfo/registry-hives> [Accessed 27 February 2024]
- [56] GANESH, Bala, 2023. *Windows Registry - Analysis and Tracking Every Windows activity*. GBHackers. Online. 3 June 2023. Available from: <https://gbhackers.com/windows-registry-analysis-tracking-everything-you-do-on-the-system/> [Accessed 27 February 2024]
- [57] DELAND, Han. *Windows registry for advanced users - Windows Server — Microsoft Learn*. Online. 26 December 2023. Available from: <https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users> [Accessed 27 February 2024]
- [58] VIA, David. *Digging Up the Past: Windows Registry Forensics Revisited*. Mandiant. Online 8 January 2019. Available from:

-
- <https://www.mandiant.com/resources/blog/digging-up-the-past-windows-registry-forensics-revisited> [Accessed 27 February 2024]
- [59] *Computer Forensics - Windows Registry - Pt. 1.* Vicarius. Online. 19 August 2022. Available from: <https://www.vicarius.io/blog/computer-forensics-windows-registry-pt-1> [Accessed 27 February 2024]
- [60] *AmCache*. Forensafe. Online. 22 April 2022. Available from: <https://forensafe.com/blogs/AmCache.html> [Accessed 27 February 2024]
- [61] *User Shell Folders* Microsoft Learn. Online. 10 September 2008. Available from: [https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc962613\(v=technet.10\)](https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc962613(v=technet.10)) [Accessed 27 February 2024]
- [62] SIMMONS, Chad. *Run and RunOnce Registry Keys - Win32 Apps*. Microsoft Learn. Online. 16 August 2023. Available from: <https://learn.microsoft.com/cs-cz/windows/win32/setupapi/run-and-runonce-registry-keys> [Accessed 27 February 2024]
- [63] *How to redirect user shell folders to a specified path by using Profile Maker*. Microsoft Support. Online. [no date]. Available from: <https://support.microsoft.com/en-au/topic/how-to-redirect-user-shell-folders-to-a-specified-path-by-using-profile-maker-ed6289ae-1f9c-b874-4e8c-20d23ea65b2e> [Accessed 27 February 2024]
- [64] *Common malware persistence mechanisms — Infosec*. Online. 13 June 2016. Available from: <https://resources.infosecinstitute.com/topics/malware-analysis/common-malware-persistence-mechanisms/> [Accessed 27 February 2024]
- [65] *Boot or logon Autostart Execution: Registry Run Keys / Startup Folder, Sub-Technique T1547.001 - Enterprise — MITRE ATT&CK®*. MITRE. Online. 2024. Available from: <https://attack.mitre.org/techniques/T1547/001/> [Accessed 27 February 2024]
- [66] CYBERMASTERV. *Just another analysis of the njRAT malware – A step-by-step approach*. Cyber Geeks. Online. 30 November 2021. Available from: <https://cybergeeks.tech/just-another-analysis-of-the-njrat-malware-a-step-by-step-approach/> [Accessed 3 March 2024]
- [67] STRATTON, Aaron. *NJRAT Malware Analysis*. InfoSec write-ups. Online. 8 October 2022. Available from: <https://infosecwriteups.com/njrat-malware-analysis-8e90dce07a9e> [Accessed 3 March 2024]

- [68] ELSHINBARY, Abdallah. *Deep analysis of Ryuk Ransomware*. Nightw0lf. Online. 5 May 2020. Available from: <https://nightw0lf.github.io/malware%20analysis/ryuk-ransomware/> [Accessed 3 March 2024]
- [69] PAMNANI, Vinay. *4657(S): A registry value was modified. - Windows 10*. Microsoft Learn. Online. 9 July 2021. <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4657> [Accessed 20 March 2024] [Accessed 20 March 2024]
- [70] *Hunting for persistence: Registry run keys / startup folder*. Cyborg Security. Online. 7 January 2021. Available from: <https://www.cyborgsecurity.com/cyborg-labs/hunting-for-persistence-registry-run-keys-startup-folder/> [Accessed 20 March 2024]
- [71] *You Can Only Hunt What You Can See: Best Endpoint Log Sources for Threat Hunting*. Cyborg Security. Online. 29 September 2020. <https://www.cyborgsecurity.com/blog/you-can-only-hunt-what-you-can-see-best-endpoint-log-sources-for-threat-hunting/> [Accessed 20 March 2024]
- [72] PAMNANI, Vinay. *4656(S, F) A handle to an object was requested. - Windows 10*. Microsoft Learn. Online. 7 September 2021. Available from: <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4656> [Accessed 20 March 2024]
- [73] WHIMS, Steve, et al. *Task Scheduler for developers - Win32 apps*. Microsoft Learn. Online. 8 February 2023. Available from: <https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page> [Accessed 23 March 2024]
- [74] DANIEL. *Scheduled task persistence*. DMFR SECURITY. Online. 7 September 2021. Available from: <https://dmfrsecurity.com/2021/09/07/scheduled-task-persistence/> [Accessed 23 March 2024]
- [75] *Persistence 101: Looking at the scheduled tasks*. stmxcsl. Online. <https://stmxcsl.com/persistence/scheduled-tasks.html> [Accessed 23 March 2024]
- [76] *Scheduled Task - Red Canary Threat Detection Report*. Red Canary. Online. 2022. <https://redcanary.com/threat-detection-report/techniques/scheduled-task/> [Accessed 23 March 2024]
- [77] WHIMS, Steve et al. *Schtasks.exe - Win32 apps*. Microsoft Learn. Online. 13 March 2023. Available from: <https://learn.microsoft.com/en-us/windows/win32/taskschd/schtasks> [Accessed 23 March 2024]

-
- [78] *Scheduled Task/Job: Scheduled Task, Sub-Technique T1053.005 - Enterprise — MITRE ATT&CK®*. Mitre. Online. 2024. Available from: <https://attack.mitre.org/techniques/T1053/005/> [Accessed 23 March 2024]
- [79] GEREND, Jason. *Set-ScheduledTask (ScheduledTasks)*. Microsoft Learn. Online. [no date]. Available from: <https://learn.microsoft.com/en-us/powershell/module/scheduledtasks/set-scheduledtask?view=windowsserver2022-ps> [Accessed 23 March 2024]
- [80] ELLAHI, Osama. *Unfolding Agent Tesla: The Art of Credentials Harvesting. Dropper analysis*. Medium. Online. 6 February 2024. Available from: <https://osamaellahi.medium.com/unfolding-agent-tesla-the-art-of-credentials-harvesting-f1a988cfd137> [Accessed 23 March 2024]
- [81] *Tarrask malware uses scheduled tasks for defense evasion*. Microsoft Security Blog. Online. 12 April 2022. <https://www.microsoft.com/en-us/security/blog/2022/04/12/tarrask-malware-uses-scheduled-tasks-for-defense-evasion/> [Accessed 27 March 2024]
- [82] PAMNANI, Vinay. *4698(S) A scheduled task was created. - Windows 10*. Microsoft Learn. Online. 7 September 2021. Available from: <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4698> [Accessed 27 March 2024]
- [83] PAMNANI, Vinay. *4688(S) A new process has been created. - Windows 10*. Microsoft Learn. Online. 24 January 2022. Available from: <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4688> [Accessed 29 March 2024]
- [84] MICHENER, John R. *Access Control: Understanding Windows file and registry permissions*. Microsoft Learn. Online. 10 September 2019. Available from: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2008/november/access-control-understanding-windows-file-and-registry-permissions> [Accessed 29 March 2024]
- [85] *Splunk - Market share, competitor insights in Security Information and Event Management (SIEM)*. 6sense. Online. [no date] <https://6sense.com/tech/security-information-and-event-management-siem/splunk-market-share> [Accessed 1 April 2024]
- [86] *Components of a Splunk Enterprise deployment*. Splunk Documentation. Online. 17 July 2018. Available from: <https://docs.splunk.com/Documentation/Splunk/9.2.0/Capacity/ComponentsofaSplunkEnterprisedeployment> [Accessed 1 April 2024]

BIBLIOGRAPHY

- [87] HARISUTHAN. *Splunk Architecture: Forwarder, indexer, and Search Head* Security Investigation. Online. 8 September 2021. Available from: <https://www.socinvestigation.com/splunk-architecture-forwarder-indexer-and-search-head/> [Accessed 1 April 2024]
- [88] *Deployment options*. Splunk Documentation. Online. 14 February 2022. Available from: <https://docs.splunk.com/Documentation/PCI/5.3.0/Install/Deploymentoptions> [Accessed 1 April 2024]
- [89] *Types of distributed deployments*. Splunk Documentation. Online. 14 September 2015. Available from: <https://docs.splunk.com/Documentation/Splunk/9.2.0/Deploy/Deploymentcharacteristics> [Accessed 1 April 2024]
- [90] *Monitor Windows event log data with Splunk Enterprise*. Splunk Documentation. Online. 21 January 2024. Available from: <https://docs.splunk.com/Documentation/Splunk/latest/Data/MonitorWindowseventlogdata> [Accessed 6 April 2024]
- [91] PAMNANI, Vinay. *Audit Registry - Windows 10*. Microsoft Learn. Online. 5 January 2021. Available from: <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/audit-registry> [Accessed 6 April 2024]
- [92] FOULDS, Iain. *Command line process auditing*. Microsoft Learn. Online. 1 May 2023. Available from: <https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/component-updates/command-line-process-auditing> [Accessed 6 April 2024]
- [93] *Getting started with alerts*. Splunk Documentation. Online. 16 April 2018. Available from: <https://docs.splunk.com/Documentation/Splunk/9.2.1/Alert/Aboutalerts> [Accessed 6 April 2024]
- [94] *Hunting for Malicious PowerShell using Script Block Logging*. Splunk Threat Research. Online. [no date]. Available from: https://www.splunk.com/en_us/blog/security/hunting-for-malicious-powershell-using-script-block-logging.html [Accessed 13 April 2024]
- [95] *What is GRR?* GRR Documentation. Online. [no date]. Available from: <https://grr-doc.readthedocs.io/en/v3.2.1/what-is-grr.html> [Accessed 15 April 2024]
- [96] *Custom alert action*. Splunk Documentation. Online. [no date]. Available from: <https://dev.splunk.com/enterprise/docs/devtools/customalertactions/> [Accessed 15 April 2024]

-
- [97] WHIMS, Steve et al. *Standard Consumer Classes - Win32 apps*. Microsoft Learn. Online. 7 January 2021. Available from: <https://learn.microsoft.com/en-us/windows/win32/wmisdk/standard-consumer-classes> [Accessed 1 May 2024]
- [98] WHIMS, Steve et al. *About WMI - Win32 apps*. Microsoft Learn. Online. 7 January 2021. Available from: <https://learn.microsoft.com/en-us/windows/win32/wmisdk/about-wmi> [Accessed 1 May 2024]
- [99] EROMOSELE, Albert. *Windows PowerShell WMI vs CIM* Medium. Online. 20 September 2023. Available from: <https://medium.com/@crawlerd01/windows-powershell-wmi-vs-cim-bd09a3469ea1> [Accessed 1 May 2024]
- [100] CHELL, Dominic. *Persistence: "the continued or prolonged existence of something": Part 3 - WMI Event Subscription*. MDSec. Online. 3 August 2020. Available from: <https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-3-wmi-event-subscription/> [Accessed 1 May 2024]
- [101] *Persistence with WMI Event Subscription and PowerShell Cradles*. Practical Security Analytics LLC. Online. 24 November 2023. Available from: <https://practicalsecurityanalytics.com/persistence-with-wmi-event-subscription-and-powershell-cradles/> [Accessed 1 May 2024]
- [102] *Business CTF 2022: Detecting and analyzing WMI Persistence - Perseverance*. Hack the Box. Online. 16 September 2022. Available from: <https://www.hackthebox.com/blog/perseverance-biz-ctf-2022-forensics-writeup> [Accessed 1 May 2024]
- [103] BARNHART, Heather Mahalik. *Finding Evil WMI Event Consumers with Disk Forensics*. SANS. Online. 22 May 2023. Available from: <https://www.sans.org/blog/finding-evil-wmi-event-consumers-with-disk-forensics/> [Accessed 1 May 2024]
- [104] *Windows Management Instrumentation - Red Canary Threat Report*. Red Canary. Online. 2024. Available from: <https://redcanary.com/threat-detection-report/techniques/windows-management-instrumentation/> [Accessed 1 May 2024]
- [105] *Persistence - WMI Event subscription*. Penetration Testing Lab. Online. 21 January 2020 Available from: <https://pentestlab.blog/2020/01/21/persistence-wmi-event-subscription/> [Accessed 1 May 2024]
- [106] PROX, Boe. *PowerShell and Events: Permanent WMI event subscriptions*. Learn Powershell — Achieve More. Online. 14 August 2013. Available from: <https://learn-powershell.net/2013/08/14/powershell->

- and-events-permanent-wmi-event-subscriptions/ [Accessed 1 May 2024]
- [107] BAČO, Ladislav. *DEF CON Hacking Conference: "Hunting for Blue Mockingbird Coinminers"*. LIFARS, a SecurityScorecard Company. Online. 8 May 2020. Available from: <https://www.lifars.com/2020/08/def-con-hacking-conference-hunting-for-blue-mockingbird-coinminers-presentation-by-ladislav-b/> [Accessed 1 May 2024]
- [108] BAČO, Ladislav. *XMRig-based CoinMiners spread by Blue Mockingbird Group*. Online. 1 June 2020. Available from: <https://www.baco.sk/posts/xmrig-blue-mockingbird/> [Accessed 1 May 2024]
- [109] <https://www.bitdefender.com/blog/businessinsights/deep-dive-into-a-fin8-attack-a-forensic-investigation/> [Accessed 1 May 2024]
- [110] KIRCH, John. *Understanding and auditing WMI*. NXLog. Online. 25 January 2022. Available from: <https://nxlog.co/news-and-blog/posts/wmi-auditing> [Accessed 1 May 2024]
- [111] *Persistence – BITS jobs*. Penetration Testing Lab. Online. 30 October 2019. Available from: <https://pentestlab.blog/2019/10/30/persistence-bits-jobs/> [Accessed 2 May 2024]
- [112] NARDELLA, Roberto. *BITS Forensics*. SANS. Online. 12 August 2019. Available from: <https://sansorg.egnyte.com/d1/0qNbxnqN2B> [Accessed 2 May 2024]
- [113] *Technique T1197 - Enterprise — MITRE ATT&CK®*. Online. 2024. Available from: <https://attack.mitre.org/techniques/T1197/> [Accessed 2 May 2024]
- [114] RAJ. *Windows Persistence using Bits Job - Hacking Articles*. Hacking Articles. Online. 17 April 2020. Available from: <https://www.hackingarticles.in/windows-persistence-using-bits-job/> [Accessed 2 May 2024]
- [115] AYASHI, Kaoru. *UBoatRAT navigates East Asia*. Palo Alto Networks Unit 42. Online. 28 November 2017. Available from: <https://unit42.paloaltonetworks.com/unit42-uboaatrat-navigates-east-asia/> [Accessed 2 May 2024]
- [116] WHIMS, Steve et al. *Managed Reference for BITS PowerShell Commands - Win32 apps*. Microsoft Learn. Online. 19 August 2020. Available from: <https://learn.microsoft.com/en-us/windows/win32/bits/bits-powershell-commands> [Accessed 2 May 2024]

- [117] *ANSSI-FR/bits_parser*. GitHub. Online. 25 January 2018. Available from: https://github.com/ANSSI-FR/bits_parser [Accessed 2 May 2024]
- [118] GEREND, Jason. *Start-BitsTransfer (BitsTransfer)*. Microsoft Learn. Online. [no date]. Available from: <https://learn.microsoft.com/en-us/powershell/module/bitstransfer/start-bitstransfer?view=windowsserver2022-ps> [Accessed 2 May 2024]
- [119] WARREN, Genevieve et al. *Introduction to Windows Service Applications - .NET Framework*. Microsoft Learn. Online. 15 September 2021. Available from: <https://learn.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications> [Accessed 3 May 2024]
- [120] JOHNSON, Jonathan. *The Defender's Guide to Windows Services* Posts by SpecterOps team members. Online. 18 January 2023. Available from: <https://posts.specterops.io/the-defenders-guide-to-windows-services-67c1711ecba7> [Accessed 3 May 2024]
- [121] *Persistence - New Service*. Penetration Testing Lab. Online 7 October 2019. Available from: <https://pentestlab.blog/2019/10/07/persistence-new-service/> [Accessed 3 May 2024]
- [122] WHEELER, Sean. *New-Service (Microsoft.PowerShell.Management) - PowerShell*. Microsoft Learn. Online. [no date]. Available from: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/new-service?view=powershell-7.4> [Accessed 3 May 2024]
- [123] *The Art of Windows Persistence*. HADESS. Online. 29 November 2023. Available from: <https://hadess.io/the-art-of-windows-persistence/> [Accessed 3 May 2024]
- [124] LAMBERT, Tony. *Blue Mockingbird activity mines Monero cryptocurrency*. Red Canary. Online. 30 April 2024. Available from: <https://redcanary.com/blog/threat-intelligence/blue-mockingbird-cryptominer/> [Accessed 3 May 2024]
- [125] LOMAN, Mark. *Ragnar Locker ransomware deploys virtual machine to dodge security*. Sophos News. Online. 21 May 2021. Available from: <https://news.sophos.com/en-us/2020/05/21/ragnar-locker-ransomware-deploys-virtual-machine-to-dodge-security/> [Accessed 8 May 2024]
- [126] *Malwarebazaar*
6e6536cc12b95070cb1a9674a4aa2c86b961bb3f4be8cae578adaa91a76898e4.

BIBLIOGRAPHY

- MalwareBazaar Database. Online. 21 December 2022. Available from: <https://bazaar.abuse.ch/sample/6e6536cc12b95070cb1a9674a4aa2c86b961bb3f4be8cae578adaa91a76898e4/#intel> [Accessed 8 May 2024]
- [127] *Malwarebazaar*
df9e900bc2aba3462d0b9d2fb4e81719604f4c63871a2225edce136c140e8fc8.
MalwareBazaar Database. Online. 8 May 2024. Available from: <https://bazaar.abuse.ch/sample/df9e900bc2aba3462d0b9d2fb4e81719604f4c63871a2225edce136c140e8fc8/> [Accessed 8 May 2024]
- [128] *Malwarebazaar*
71717c434a87554cdeaf52f6c6f3049968ddb2f74f8d0386938977b86e827fc2.
MalwareBazaar Database. Online. 6 May 2024. Available from: <https://bazaar.abuse.ch/sample/71717c434a87554cdeaf52f6c6f3049968ddb2f74f8d0386938977b86e827fc2/> [Accessed 8 May 2024]
- [129] *Malwarebazaar*
06ca6e79b1e98c0d2223781294b4663da9d8e31d0d4e0a0528058fe74865db24.
MalwareBazaar Database. Online. 8 May 2024. Available from: <https://bazaar.abuse.ch/sample/06ca6e79b1e98c0d2223781294b4663da9d8e31d0d4e0a0528058fe74865db24/> [Accessed 8 May 2024]
- [130] *Malwarebazaar*
f4bb0089dcf3629b1570fda839ef2f06c29cbf846c5134755d22d419015c8bd2.
MalwareBazaar Database. Online. 6 February 2024. Available from: <https://bazaar.abuse.ch/sample/f4bb0089dcf3629b1570fda839ef2f06c29cbf846c5134755d22d419015c8bd2/> [Accessed 8 May 2024]
- [131] *Protocol Buffers*. Google LLC. Online. 2024. Available from: <https://protobuf.dev/> [Accessed 8 May 2024]

Contents of attachments

The attachment contains scripts and configuration files referenced in the text. It contains two main directories, one containing the content for the Windows endpoint instance (configuraion and testing scripts) and one containing the content for the splunk-server (configuration files, whitelists, alert action scripts and saved detection rules). No cusotm files were deployed on the GRR server.

readme.txt	the file with archive contents description
splunk-server	contains custom files used on the splunk server
├─ grr_handling	contains the source of the custom alert actinon
├─ whitelists	contains whitelists for selected rules
├─ savedsearches.conf ..	configuration file containing all the detection rules
windows endpoint	contains custom files used on the splunk server
├─ scripts	contains test and configuration scripts for implemented persistence techniques
│ ├─ BITS	test scripts for BITS jobs
│ ├─ Registry ..	test and configuration scripts for registry and startup folder
│ ├─ Services	test scripts for Windows Services
│ ├─ Schtasks	test scripts for Scheduled tasks
│ ├─ WMI	test scripts for WMI event subscription
├─ fleetspeak-client.config	configuration of the GRR agent
├─ inputs.conf	configuration of log sources sent to Splunk server
├─ outputs.conf	configuration of Splunk connection
├─ thesis	the directory of L ^A T _E X source codes of the thesis
text	the thesis source directory
└─ thesis.pdf	the thesis in PDF format