

TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL & COMPUTER ENGINEERING



DIPLOMA THESIS

Leveraging Infrastructure-as-Code for Automated Malware Analysis and Intelligence Collection

Author:

Christos Fotopoulos

Committee:

Ioannis Marinakis , Professor
Sotiris Ioannidis, Professor
Magadlini Marinaki, Researcher

Date

Abstract

In today's digital age, Internet and computers are being used daily by both individuals and companies. Malicious actors are attempting to exploit this reliance by trying to gain unauthorized access to devices. An important tool in their arsenal is malicious software, commonly known as malware, which infects computers to perform tasks such as stealing sensitive data and destroying the availability of the system. To combat these threats of cyber attacks and malware infections effectively, Cyber Threat Intelligence is needed. This intelligence can provide information to potential victims and thus proactively prepare them.

This thesis presented a novel approach for acquiring intelligence, emphasizing not the ingestion of existing reports but the proactive creation of new intelligence through automated dynamic malware analysis. To achieve this, a workflow and a proof-of-concept tool were created to help with the automated execution of malware and the extraction of the created artifacts.

The tool drew inspiration from traditional sandboxes, but instead of performing typical malware analysis, it focused on bulk investigation and easy customization. More specifically, by heavily focusing on automation and pioneering concepts such as infrastructure as code (IaC), it allowed the analysis of multiple malware simultaneously. On top of that, using configuration management tools, such as Ansible, it enabled users to easily extract new artifacts by creating scripts with their desired language or tool.

During the analysis phase, the collected artifacts were used to create intelligence in the form of Indication of Compromise (IOC), Tactics, Techniques, and Procedures (TTPs), Fingerprints, and Detections using Detection as Code (DaC) tools such as Sigma. Due to the nature of the approach, we managed to create intelligence from a large number of malware samples instead of analyzing their source code, which would have been a more time-consuming approach.

Acknowledgements

I would like to thank the Technical University of Crete for its support in completing this thesis. Specifically, I would like to thank Eva Papadogiannaki for helping me during the research and the writing of the Thesis. Furthermore, I would like to thank Ioannis Marinakis, Sotiris Ioannidis, and Magdalini Marinaki, who were the members of the thesis examination committee.

Contents

List of Figures	5
1 Introduction	7
1.1 Purpose and Objectives	7
1.2 Contributions	8
1.3 Structure of Thesis	8
2 Background	10
2.1 Core Concepts in Malware Analysis	10
2.1.1 Dynamic Malware Analysis	10
2.1.2 Sandbox	10
2.1.3 Malware Families	10
2.1.4 Command and Control	11
2.1.5 Indicators of Compromise	12
2.1.6 Tactics, Techniques, and Procedures	12
2.1.7 Detections	12
2.2 Automation Tools	14
2.2.1 Proxmox	14
2.2.2 Terraform	14
2.2.3 Ansible	15
2.3 Analysis Tools	15
2.3.1 JA4+ Fingerprints	15
2.3.2 Sysmon	18
2.3.3 Malware Bazaar	18
2.3.4 Censys Search	18
2.3.5 SQLite Database	18
3 Related Work	19
4 Methodology	21
4.1 System Design	22
4.1.1 Analysis Network	22
4.1.2 Isolate Network	22
4.1.3 Intermediate Devices	22
4.2 Development	22
4.2.1 Sandbox Service	23
4.2.2 Deploying Windows Sandboxes	24
4.2.3 Preparing Information for Ansible	25

4.2.4	Ansible Playbooks for Configuring the Sandbox	26
4.2.5	Execution of Malware Samples	28
4.2.6	Artifact Collection	29
4.2.7	Collectors and Data Storage	30
5	Findings	35
5.1	Data Cleaning	35
5.1.1	Defective Samples	35
5.2	Malware Insights	40
5.2.1	Agent Tesla Insights	40
5.2.2	Redline Insights	46
5.2.3	Remcos Insights	51
5.2.4	Similarity Analysis Across Malware Families	56
5.2.5	Comparing Benign and Malicious JA4+ Fingerprints	57
5.3	Malware Detections	57
5.3.1	Endpoint Detections	57
5.3.2	Network Detections	64
5.3.3	JA4+ Detections	69
6	Discussion	71
6.1	Limitations	71
6.2	Future Work	71
7	Conclusion	73
	Bibliography	74

List of Figures

2.1	Suricata Rule Sample [0.1em]Source: https://docs.suricata.io/en/latest/rules/intro.html .	12
2.2	Sigma Rule Sample [0.1em]Source: https://sigmahq.io/docs/basics/rules.html	13
2.3	Terraform Template Sample [0.1em]Source: https://registry.terraform.io/providers/hashicorp/vsphere/latest/docs/resources/vsphere-vm-template.html	14
2.4	Ansible Playbook Sample [0.1em]Source: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html	15
2.5	TLS Handshake [0.1em]Source: https://www.cloudflare.com/en-gb/learning/ssl/what-happens-in-a-tls-handshake	16
2.6	JA4 Fingerprint [0.1em]Source: https://blog.foxio.io/ja4+-network-fingerprinting	16
2.7	JA4S Fingerprint [0.1em]Source: https://blog.foxio.io/ja4+-network-fingerprinting	17
2.8	JA4X Fingerprint [0.1em]Source: https://blog.foxio.io/ja4+-network-fingerprinting	17
4.1	Network Topology	21
4.2	Methodology	23
4.3	Sandbox Service Flowchart	24
4.4	proxmox_automation Flowchart	25
4.5	Program Flow	30
4.6	JA4 Collector Flowchart	32
4.7	Domain/SNI Collector Flowchart	33
5.1	Non-valuable Agent Tesla Samples from Endpoint logs	36
5.2	Non-valuable Redline Samples from Endpoint logs	37
5.3	Non-valuable Remcos Samples from Endpoint logs	37
5.4	Non-valuable Agent Tesla from the dataset	38
5.5	Non-valuable Redline from the dataset	39
5.6	Non-valuable Remcos from the dataset	39
5.7	Agent Tesla - Windows Defender Exclusion	40
5.8	Agent Tesla - Process Hollowing	41
5.9	Agent Tesla - Detected Samples	42
5.10	Agent Tesla FTP C2 Communication	43
5.11	Agent Tesla HTTP C2 Communication	43
5.12	Anyrun for Agent Tesla	44
5.13	Redline - Windows Defender Exclusion	46
5.14	Redline - Process Hollowing	46
5.15	Redline Suspicious SNI	48
5.16	Redline First SOAP Variant[18, 19]	49
5.17	Redline Second SOAP Variant[24]	49
5.18	Redline Certificate	51

5.19 Remcos - Windows Defender Exclusion	52
5.20 Remcos - Credential Dumping with Nirsoft Tools	52
5.21 Remcos - Detected Samples	53
5.22 Remcos Client Unencrypted Traffic	54
5.23 Remcos Server Unencrypted Traffic	54

Chapter 1

Introduction

Computers, smartphones, and Internet of Things (IoT) devices are a big part of human life. More specifically, individuals use the Internet daily for entertainment and communication. Computers are also being utilized at the enterprise level for multiple reasons, such as advertisement, communication, and performing challenging tasks for humans.

Due to their rise in popularity, there have also been multiple malicious actors trying to exploit this phenomenon to their own advantage. In recent years, cyber-attacks have been significantly increasing, thus making them an ascending threat to businesses. Cybercriminals, after compromising a company's infrastructure, can use it for malicious purposes, exfiltrate sensitive data, and encrypt important files, which may result in the loss of essential data and business continuity. The scope of such actors has also shifted from targeting big mature companies to smaller businesses and individuals, which may be an easier target. Based on the above, it can be concluded that cyber attacks are an important threat to everyone and should not be ignored. [32, 29]

To stop such attacks, individuals and companies must use a comprehensive approach that combines security practices, tools, and trained personnel. Additionally, to enhance its security posture, a company should consume and create threat intelligence to stay up to date with any threads that may be directed at it.

1.1 Purpose and Objectives

This research aims to bring light to a different approach to creating intelligence through analyzing malicious software. More specifically, it aimed to take advantage of the benefits of moving away from performing advanced techniques such as reverse engineering. Instead, it focused on automated execution of the samples and extraction of the artifacts. By doing so, an organization could create intelligence or perform evaluations on security products such as an Antivirus, an Endpoint Detection and Response (EDR), or even a detection on a security information and event management (SIEM) platform.

Although implementing this methodology can have multiple use cases, the focus of the research was on creating intelligence by automated collection of artifacts from the three most used malware families at the time: Agent Tesla, Redline, and RemcosRAT. Finally, after the automated workflow has finished, manual analysis will be performed on the collected data to clean it and perform the intelligence analysis.

The aforementioned process can quickly provide important insights into malware families' techniques, provide IOCs that security products can monitor and block, and help create detections at the endpoint and packet levels. This information can be extremely valuable for enhancing a company's security.

1.2 Contributions

With this approach, our objective was to delve into threat intelligence creation using an automated approach to malware analysis using sandboxes. The following contributions were made:

- It showcased a different approach to ingesting threat intelligence. Instead of trying to optimize the analysis of free and commercial CTI reports using tools such as large language models and AI, we displayed the creation of intelligence through automated malware analysis using sandboxes. By following this workflow, the problems of analyzing expensive dissimilar reports could be avoided.
- Furthermore, the thesis presented a different approach to traditional state-of-the-art sandboxes. An on-premises sandbox was created following infrastructure as code principles and automation practices. The sandbox's use was shifted from determining if a file was malicious to analyzing known malware for intelligence. The aforementioned modifications allowed the analysis of multiple samples simultaneously and supported easy customization. Finally, this approach could also allow the bulk analysis of classified malware that could not be uploaded to commercial sandboxes.
- Furthermore, the thesis illustrated that the ability to analyze large malware datasets efficiently, thus significantly facilitated the creation of threat intelligence and an understanding of the malware's behavior. More specifically, the behavior of Agent Tesla, Redline, and Remcos malware families was investigated. By researching them, their Tactics, Techniques, and Procedures were identified, and detections were created on the endpoint and network level.

1.3 Structure of Thesis

- **Chapter 1:** The first chapter briefly introduces a non traditional way of creating intelligence and a new approach to Sandboxes and malware analysis, focusing more on speed, efficiency, and scalability than in-depth analysis.
- **Chapter 2:** The second chapter provides the necessary background to understand the technologies used to analyze the malware samples, extract the collected artifacts, and create the final intelligence.
- **Chapter 3:** In this section, similar research and papers are compared to emphasize the advantages and disadvantages of this methodology compared with the commonly used techniques.
- **Chapter 4:** This segment describes the custom proof-of-concept tool that was created and the infrastructure needed to use it.
- **Chapter 5:** In this chapter, the collected information is evaluated after the proper filtering and cleaning.

- **Chapter 6:** Finally, this section concludes with a discussion of the limitations of this research and possible further improvements to the feature.

Chapter 2

Background

This chapter presents the background concepts required for this thesis. It also briefly introduces the tools used to create the automation engine and to analyze the collected artifacts.

2.1 Core Concepts in Malware Analysis

2.1.1 Dynamic Malware Analysis

According to Crowdstrike, Malware analysis is the process of understanding the behavior and purpose of a suspicious file or URL. The output of the analysis aids in the detection and mitigation of the potential threat. Depending on the methods used for the analysis, it can be divided into two categories: Dynamic Analysis and Static Analysis.[33]

During the Dynamic Analysis, the suspected malicious sample is executed in a safe environment, and its actions are analyzed. On the other hand, during the Static Analysis, the sample is not executed, but analyzed for malicious components.

2.1.2 Sandbox

A malware sandbox is a virtual environment that isolates and analyzes potentially malicious software's behavior. It replicates a standard operating environment, such as Windows, where samples can be executed without risk to actual systems. Sandboxes typically have an analysis engine that performs a deep investigation on the sample, trying to extract IOCs and deduct a signature related to its behavior. As discussed in this thesis, a different approach has been taken, focusing more on the speed, the number of samples, and accessibility to new artifacts that can be collected, thus focusing more on threat intelligence than malware analysis.

2.1.3 Malware Families

A malware is malicious software that performs harmful activities such as damaging a computer, gaining unauthorized access, or exfiltrating sensitive data. A malware family is a group of malware with similar characteristics, such as code, behavior, and attacks. In this work, we will analyze the malware families: Agent Tesla information stealers, the Redline information stealers, and RemcosRAT (Remote Access Trojans).

Agent Tesla is malware as a service written on .NET. It functions both as a Remote Access Trojan (RAT) and an infostealer. More specifically, it offers various functionalities, such as Keylogging, taking screenshots, and stealing browser passwords. It can use many protocols for data exfiltration and to receive commands from malicious actors, most commonly HTTP and SMTP. The primary method of distribution is phishing emails containing URLs or PDF files containing links. After the user is tricked, he will be prompted to download the Agent Tesla malware.[28]

Redline, first observed in 2020, is another malware as a service, meaning malware developers sell malware builders and their infrastructure, thus enabling the easy execution of cyberattacks. Similarly to AgentTesla, it has Infostealing and RAT functionalities. On the other hand, since it lacks an out-of-the-box distribution method, many infections originate from phishing email software and direct messaging on social media platforms. A new malware called MetaStealer was also built on top of the Redline code, making its first appearance in 2022[23].

Remcos is a Remote Administration Tool (RAT) created by the Germany-based security company Breaking-Security but adopted by many malicious actors. It is a remote control and surveillance tool, usually accompanied by dropper malware that executes it in memory. The tool has been used by malicious actors since its first version was published on July 21, 2016, and even the newer versions have been identified in malicious campaigns[7].

2.1.4 Command and Control

According to MITRE, Command and Control (C2) consists of techniques that adversaries may use to communicate with systems under their control within a victim network. A significant part of this research was analyzing the C2 communications and the behavior of the C2 servers[6, 14].

The process of malware periodically communicating with its C2 server is called beaconing. Due to the excessive and periodic requests that the malware performs, it has been an excellent detection opportunity for network security systems. In response to the evolution of network security systems, malware developers, penetration testers, and malicious actors were forced to adapt and use commonly used protocols, thus hiding in the legitimate noise the user or the operating system performs.

One of the most common protocols for both legitimate users and malware is HTTP. HTTP (Hypertext Transfer Protocol) is the foundation of data communication for the World Wide Web and is used when regular users browse a legitimate site. Malicious actors leverage this to send and receive commands, thus hiding in the excessive amount of traffic generated daily. For security reasons, the HTTPS (Hypertext Transfer Protocol Secure) protocol was later created, offering data encryption using SSL/TLS. Although this was an important step in allowing users to browse safely, it enabled malicious actors to hide their activities more efficiently[12].

Threat actors can also use other protocols for communication, such as DNS that translates domains to IP addresses. This protocol can be weaponized by adding data to its payload or as a subdomain, which is usually ignored by security tools. FTP (File Transfer Protocol) helps to transfer files between a client and a server over a network, and SMTP (Simple Mail Transfer Protocol) allows data to be transferred via email. They both can be utilized to exfiltrate large amounts of sensitive data. Finally, it is worth mentioning that malware still uses custom evasive and encrypted protocols to perform Command and control communication.

2.1.5 Indicators of Compromise

An indicator of compromise (IOC) is evidence that someone may have breached an organization's network or endpoint. IOCs can be obtained from multiple artifacts, such as malicious IP addresses that are being used as C2, SHA256, or MD5 file hashes representing malicious files or even file names. Although IOCs provide high-confidence detection of an infection, they can be easily changed by malicious actors. For example, when adding an extra line of code, the SHA256 or MD5 hash of the sample will change.

2.1.6 Tactics, Techniques, and Procedures

Tactics, Techniques, and Procedures (TTP) describe the behavior of a threat actor and is a structured framework for executing a cyberattack. By identifying an attacker's TTPs, an organization can take security measures, such as creating detections for them, thus reducing the risk of compromise. An important tool for identifying TTPs is MITRE ATT&CK, a knowledge base of adversarial techniques based on real-world observations. This tool provides valuable information regarding TTPs, as well as which Advanced Persistence Threat (APT) Groups are using them. [6]

2.1.7 Detections

In cybersecurity, detection refers to the combination of conditions needed to identify, notify, and take action regarding suspicious activity.

For example, when a user visits a webpage using the program `calc.exe` (calculator), create an alert and close the connection.

- Condition 1: Process `calc.exe` being executed
- Condition 2: Request to a webpage
- Notify: Create an alert
- Action: Close the connection

Suricata Rule

Suricata is an open-source Intrusion Detection System (IDS) and Intrusion Prevention System (IPS). Suricata rules are a set of instructions that defines criteria and patterns in network traffic and what action to take if they are detected. Thus, it can be utilized to create detections for anomaly and suspicious patterns in malware traffic. Creating such detections can later allow an organization to detect activities in its own network.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET Request  
Containing Rule in URI"; flow:established,to_server; http.method;  
content:"GET"; http.uri; content:"rule"; fast_pattern; classtype:bad-  
unknown; sid:123; rev:1;)
```

2.1: Suricata Rule Sample

Source: <https://docs.suricata.io/en/latest/rules/intro.html>

As it can be observed from the aforementioned example 2.1 a suricata rule consists of the following:

- The action: Determining the outcomes that occur when the rule is triggered.
- The header: Defining the protocol, IP address, or the traffic direction, ports, and direction of the rule.
- The rule options: Defines the details of the rule. In this section, fields such as HTTP URL or bytes inside the packet can be signatures.

Sigma Detections

Sigma is a signature format that describes relevant log events. In other words, it is used the same way Suricata was used, but instead of investigating network-related activity, it analyzes endpoint logs. Although endpoint logs can provide more information, due to their significant number, they can be size-consuming therefore a combination of the two detection methods is important.

```
1 # ./rules/cloud/okta/okta_user_account_locked_out.yml
2 title: Okta User Account Locked Out
3 id: 14701da0-4b0f-4ee6-9c95-2ffb4e73bb9a
4 status: test
5 description: Detects when a user account is locked out.
6 references:
7   - https://developer.okta.com/docs/reference/api/system-log/
8   - https://developer.okta.com/docs/reference/api/event-types/
9 author: Austin Songer @austinsonger
10 date: 2021-09-12
11 modified: 2022-10-09
12 tags:
13   - attack.impact
14 logsource:
15   product: okta
16   service: okta
17 detection:
18   selection:
19     displaymessage: Max sign in attempts exceeded
20   condition: selection
21 falsepositives:
22   - Unknown
23 level: medium
```

2.2: Sigma Rule Sample

Source: <https://sigmahq.io/docs/basics/rules.html>

Creating Suricata and Sigma rules is an important result of a malware research since it can allow an organization to take security measures before a cyber attack occurs. It is also worth mentioning that choosing a fast and large-scale analysis instead of a deep dive into the malware samples allowed the creation of multiple detections quickly.

2.2 Automation Tools

This research and the created workflow heavily utilized automation tools to create intelligence. Firstly, the automated deployment of Virtual Machines was an important aspect of creating the new Windows sandboxes to execute the malware. In addition, the virtual machines needed to be automatically configured, the malware executed, and the artifacts collected.

2.2.1 Proxmox

Proxmox VE (Virtual Environment) is an open-source type one hypervisor used to deploy and oversee virtual machines and LXC containers. It is a robust and flexible virtualization solution for virtualizing OSes such as Windows and Ubuntu servers and using them as standalone computers[11].

2.2.2 Terraform

Terraform is an Infrastructure as Code (IaC) tool that assists in building, changing, and versioning cloud and on-prem resources. Its primary purpose was to provide Proxmox with information regarding the resources a virtual machine should have (such as CPU, memory, and network card) and, in combination with Proxmox, automatically deploy and destroy virtual machines [4].

```
data "vsphere_datacenter" "datacenter" {
  name = "dc-01"
}

data "vsphere_datastore" "datastore" {
  name           = "datastore-01"
  datacenter_id = data.vsphere_datacenter.datacenter.id
}

data "vsphere_compute_cluster" "cluster" {
  name           = "cluster-01"
  datacenter_id = data.vsphere_datacenter.datacenter.id
}

data "vsphere_network" "network" {
  name           = "VM Network"
  datacenter_id = data.vsphere_datacenter.datacenter.id
}

resource "vsphere_virtual_machine" "vm" {
  name             = "foo"
  resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
  datastore_id     = data.vsphere_datastore.datastore.id
  num_cpus        = 1
  memory          = 1024
  guest_id        = "otherLinux64Guest"
  network_interface {
    network_id = data.vsphere_network.network.id
  }
  disk {
    label = "Hard Disk 1"
    size  = 20
  }
}
```

2.3: Terraform Template Sample

Source:

https://registry.terraform.io/providers/hashicorp/vsphere/latest/docs/resources/virtual_machine.html

2.2.3 Ansible

Ansible is an open-source, command-line IT automation software application used to configure systems, deploy software, and orchestrate advanced workflows. It uses files similar to Terraform, called Ansible Playbooks, allowing easy customization. This software can perform simultaneous changes on multiple hosts by connecting with them using the Secure Shell (SSH) or Windows Remote Management (WINRM) protocols [2].

```
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest

    - name: Write the apache config file
      ansible.builtin.template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
    - name: Ensure postgresql is at the latest version
      ansible.builtin.yum:
        name: postgresql
        state: latest

    - name: Ensure that postgresql is started
      ansible.builtin.service:
        name: postgresql
        state: started
```

2.4: Ansible Playbook Sample

Source: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html

2.3 Analysis Tools

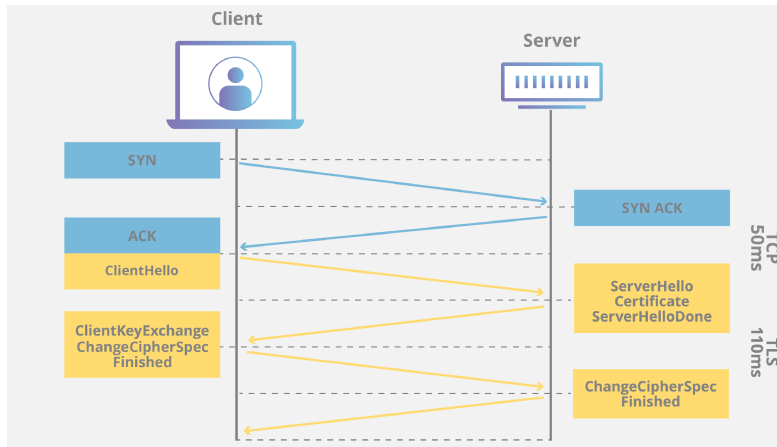
In the analysis tools category, mechanisms for creating artifacts, enriching them with information, and helping to create intelligence were added. These tools were utilized to create logs and fingerprints and enrich the collected data with information from online services. After the automated workflows had concluded, their final use was to assist in analyzing the artifacts and creating intelligence, which is the goal of the research.

2.3.1 JA4+ Fingerprints

JA4+ is a suite of network fingerprinting methods utilized for tasks such as scanning for threat actors, malware detection, and location tracking. The fingerprints used are the TLS Client Fingerprint JA4, the TLS Server Fingerprint JA4S, and the X509 TLS Certificate Fingerprinting JA4X[21, 22].

Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network, such as the Internet. It is used to encrypt most standard protocols, such as HTTPS, providing not only privacy but security to man in the middle of attacks that may try to intercept and manipulate a user's traffic. At the beginning of a TLS connection, a handshake has to be made, where the client sends a Client Hello message, and the

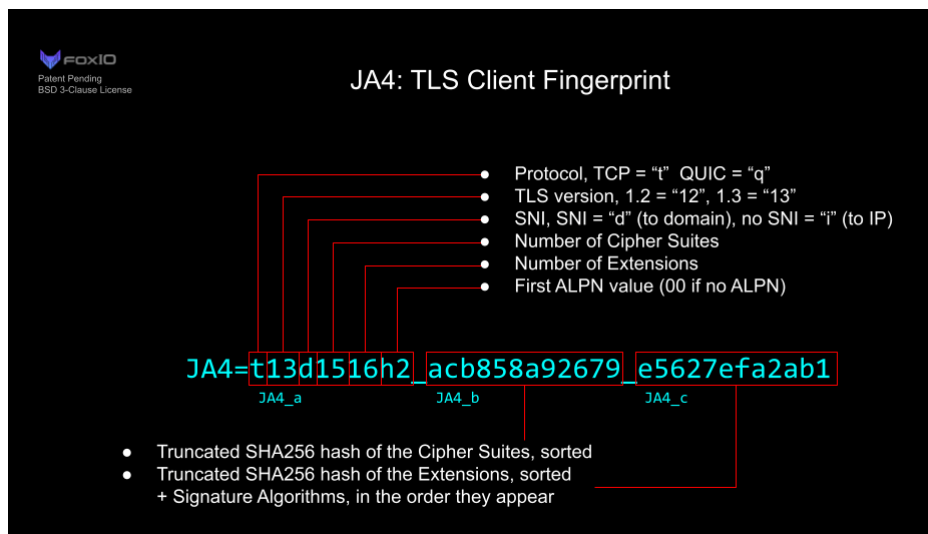
server responds with a Server Hello message. At the same time that the server sends its hello packet, it may send its certificates, which contain the website's public key, identity, and related information[10, 8, 9].



2.5: TLS Handshake

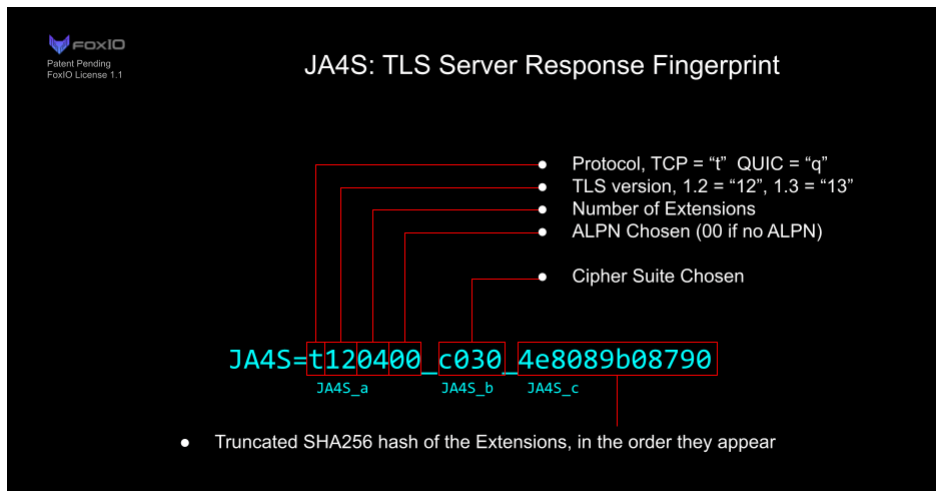
Source: <https://www.cloudflare.com/en-gb/learning/ssl/what-happens-in-a-tls-handshake>

JA4 or TLS Client Fingerprint is a human-readable fingerprint that contains information that can distinguish a TLS handshake and make it unique. In contrast to the JA4 that collects its information from the Client Hello part of the handshake, JA4S or TLS Server Fingerprint queries information from the Server Hello packet.



2.6: JA4 Fingerprint

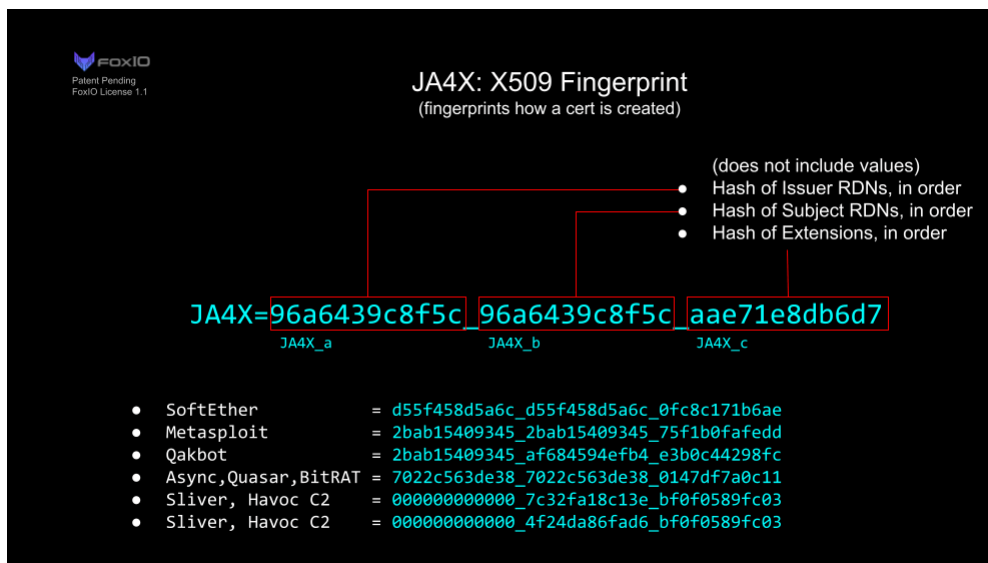
Source: <https://blog.foxio.io/ja4+-network-fingerprinting>



2.7: JA4S Fingerprint

Source: <https://blog.foxio.io/ja4+-network-fingerprinting>

JA4X or X509 TLS Certificate Fingerprint, fingerprints the way in which TLS certificates are generated — not the values within the certificate. More specifically it will identify the order of the certificates and not their actual values. This can identify applications and settings used to create the certificate which can be extremely useful in threat hunting as threat actors will create different certificates but tend to use the same methods to create said certificates, thereby having the same JA4X fingerprint.



2.8: JA4X Fingerprint

Source: <https://blog.foxio.io/ja4+-network-fingerprinting>

2.3.2 Sysmon

System Monitor (Sysmon) is a Windows system service and device driver that, once installed on a system, remains resident across system reboots to monitor and log system activity to the Windows event log. Installing it can provide fruitful information for the activity performed by the processes on the endpoint[30].

2.3.3 Malware Bazaar

MalwareBazaar is a project from abuse.ch that aims to share malware samples with the information security community, AV vendors, and threat intelligence providers. Its collection is updated daily by abuse.ch and security researchers, thus providing still relevant samples. Another great feature is that it categorizes each malware sample with tags and thus provides information such as its family and architecture[5].

2.3.4 Censys Search

Censys is a web-based search platform that identifies Internet-connected assets. During the analysis phase, it was used to provide information regarding the external services hosted on the malicious servers[3].

2.3.5 SQLite Database

SQLite is a server-less relational database management system. It has two important benefits in comparison with other databases. Firstly, it does not require a server installation or complex configuration. On top of the second feature is portability, the database is stored in a single file and thus can be effortlessly transferred between workstations.

Chapter 3

Related Work

With the increasing number of cyber attacks, there has been an apparent demand for ways to establish proactive defense mechanisms for such threats. To achieve such a task, intelligence is needed to specify the threat actors that can target an organization, their techniques, and their indicators of compromise. Gathering such intelligence is a challenging task, and it is one of the key areas currently being researched.

One of the main difficulties is the format of the intelligence and its accuracy. Free and open-source feeds can vary widely in nature for example, they may be articles, podcasts, or other content types. The way information is structured can vary significantly. On top of that feeds may also include low-quality artifacts or false positives thus decreasing their value. A research by Bouwman et al. [37] called "A different cup of TI? The added value of commercial threat intelligence" showed that the above problems can be solved by Commercial Threat Intelligence, which is consistent and offers high fidelity but can be very expensive. Furthermore, during their research, they discovered that indicators from different feeds rarely overlap. Therefore, defenders must invest in multiple commercial feeds to get a holistic view of them.

On the other hand, research performed by Alam et al. [36] tackled the problem from a different approach. In the research, the framework LADDER was created, which aims to automatically extract phases of an attack and map them to the MITRE framework. By doing so, the opportunity to automatically analyze a large number of reports was created, providing a lot of intelligence and techniques used by threat actors.

This paper was inspired by the above research, and especially the inclusion of automation in intelligence collection. It is important to note that this research focused on creating new intelligence through malware analysis rather than parsing intelligence from reports. To be more precise, throughout this research, modifications were made to the cybersecurity concept of Sandboxes. More specifically, rather than focusing on determining whether a program is malicious or not, the approach concentrated on scalability and the analysis of multiple known malicious samples. On top of that, in contrast to traditional sandboxes, this approach also added the concept of high customization, thus enabling the easy collection of new artifacts.

Djenna et al. [38], in the paper "Artificial Intelligence-Based Malware Detection, Analysis, and Mitigation", also addressed the growing malware issue. To do so, they tackle the problem by performing automated malware analysis using Artificial Intelligence. Specifically, they researched

malware classification by performing in-depth analysis using Artificial Intelligence and Machine Learning models. On the other hand, rather than determining whether a program is malicious, this research confronted the problem from a different perspective. It proposed a high-level automated dynamic analysis to extract the behavior from confirmed malicious programs. The intelligence gathered from this analysis can enhance security measures in various ways, such as blocking the collected IOCs and creating detections based on them.

In their research “Dynamic Malware analysis Using Cuckoo Sandbox”, Jamalpur et al.[39] analyzed a malicious sample using the Cuckoo Sandbox and the disassembly of the IDA Pro tool. Cuckoo is an open-source automated malware analysis system that automatically executes and analyzes files in an isolated operating system. IDA Pro Disassembler is a tool that translates machine language into assembly language, thus allowing an analyst to understand the functionality of a compiled binary. Although both methods provided detailed information about the malware’s behavior, analyzing it with Cuckoo took significantly less time, thus highlighting the importance of automated analysis. The difference between this research and the one by the two researchers is the different ways of using a Sandbox tool. More specifically, this research showcases an alternative approach, aiming at the automated analysis for intelligence collection, unlike the other research that focuses on analyzing samples to clarify if they are malicious.

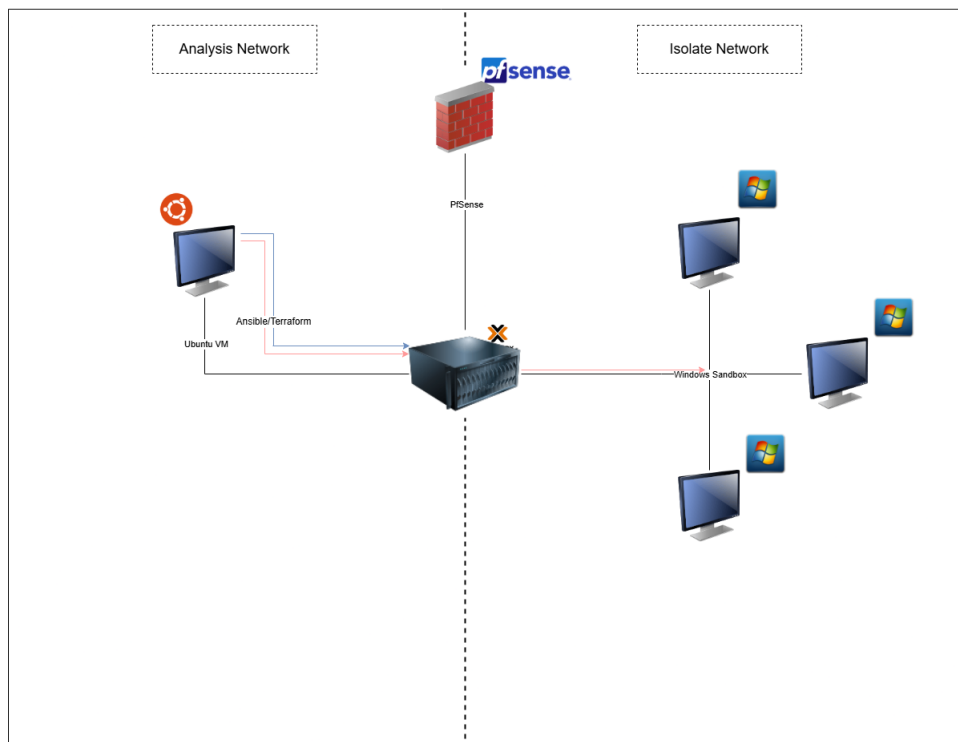
The research paper “Redefining Malware Sandboxing: Enhancing Analysis Through Sysmon and ELK Integration” by Mahmoud et al. [40] contained many similarities with this research. Both of them proposed the execution of malware samples in automatically deployed Windows Virtual Machines. On top of that, the enhancement of endpoint data with Sysmon was also utilized. The main difference is that this research focused on creating an automated workflow that provided more customization. The study performed by the aforementioned two researchers concentrated more on the sandbox by utilizing tools such as PAFish. PAFish is a testing tool that employs various techniques to detect virtual machines. It can make a sandbox environment resemble a workstation, effectively deceiving malware. Finally, due to the tool’s high customization, it was easy to analyze recently discovered samples utilizing multiple data sources, contrary to the second research, which used a malware dataset by VxUnderground with only endpoint logs.

In conclusion, our research showcased a novel approach to creating intelligence utilizing malware analysis and malware sandboxes. Instead of performing in-depth analysis of programs, such as disassembling, debugging, and scanning them with AI algorithms to understand if they are malicious, the focus was shifted to the intelligence collection. Therefore automated dynamic analysis of multiple known malicious samples was performed. The purpose was to extract valuable intelligence that can later be used to enhance the understanding of malware families and their behavior, collect malicious IOCs, identify malware infrastructure, and create custom detections. To achieve this, changes were made to traditional sandboxes to improve their scalability and ease of customization.

Chapter 4

Methodology

The network diagram of this lab is presented in Figure 4.1. As noted, the topology consists of two networks: the analysis network containing the orchestrator devices and the isolated network containing temporary Windows virtual machines infected with malware.



4.1: Network Topology

4.1 System Design

During this section, a small introduction was made regarding the infrastructure's design and the devices used. On top of that, the role of each device was described, and the OS versions that were used.

4.1.1 Analysis Network

Ubuntu Orchestrator

The Ubuntu Orchestrator Virtual Machine (Ubuntu 22.04.5 LTS) contained the automation application and was used to command the Hypervisor to create Windows Virtual Machines. This was achieved using Terraform to deploy the virtual machines and Ansible to configure them and perform the commands on the hosts. Finally, it stored the collected artifacts and hosted the SQLite database to analyze them.

4.1.2 Isolate Network

Windows 10 Sandboxes

Inside the isolated network, Windows 10 machines were automatically created. After they were configured using Ansible, malware samples were executed on machines, and then they were destroyed. This process was repeated until all the samples have been analyzed.

4.1.3 Intermediate Devices

PfSense Firewall

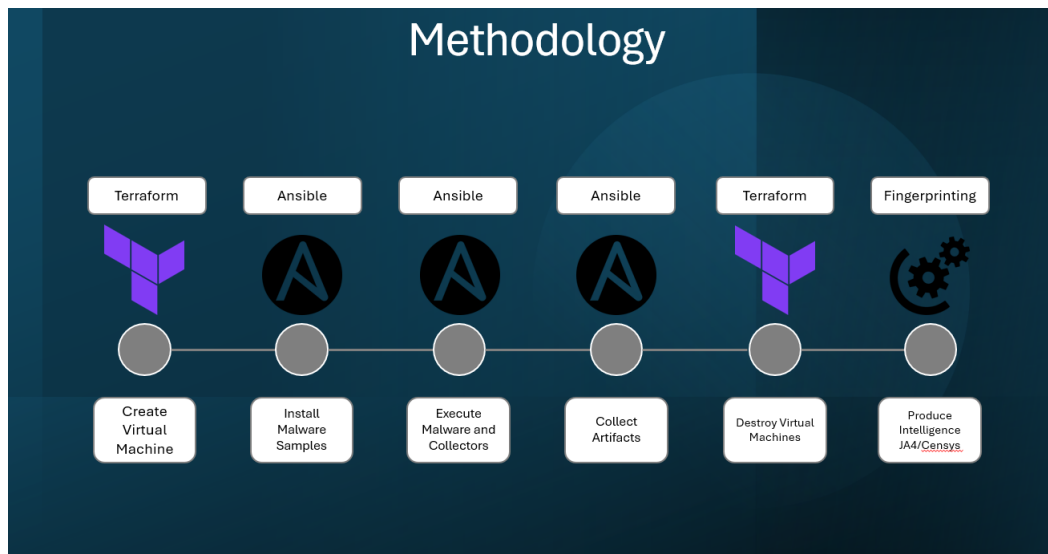
PfSense (2.7.0-RELEASE amd64) is an open-source firewall and router with many features, such as threat management, load balancing, and multi-WAN. This device was critical for this research since it separates the Ubuntu workstation from the infected devices, thus making the analysis of the malware safer.

Proxmox Hypervisor

Proxmox is a type one hypervisor on which Ubuntu, pfSense, and Windows virtual machines are created. The Proxmox used in this version was 7.2-3 and the Kernel was 5.15.30-2. It is also worth mentioning that Proxmox had two VLANs, one for each network.

4.2 Development

This work presented a methodology consisting of the automation of the following three parts: (i) the creation of the Windows environment, (ii) the execution of the malware, and (iii) the collection of the artifacts. Specifically, in this work, we also analyzed the collected artifacts to create intelligence.



4.2: Methodology

4.2.1 Sandbox Service

The Python tool `sandbox.service.py` was created to run all of the automation programs in the correct order and with the proper logging. For it to run, the name of the malware family was required as an argument, as well as the `malware_list.txt` file, which contained the file hashes of the samples that had to be analyzed.

The tool's logic consisted of an infinite loop that executed the Terraform and Ansible Playbooks until no more samples were found during the malware installation phase. In that case, the variable `end_service` was set to `False`, and the program stopped.

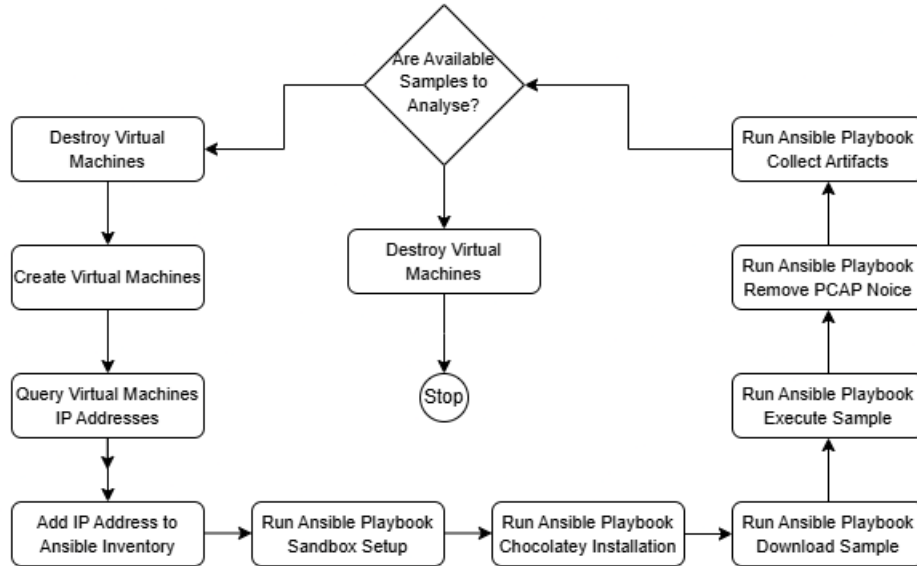
Firstly, when the `sandbox.service.py` was executed, it deprovisioned all the virtual machines managed by the Terraform configuration. This was done at the start of each loop (and when the program exited) to account for crashes and unexpected behaviors that may keep machines from being destroyed. Therefore, the scenario of deploying an already deployed machine will be avoided.

After that, the tool performed the automation workflow that will be later discussed, with the addition of logging. More specifically, a text was printed to the console whenever a task was completed, and from the other side, when a task was unsuccessful, a log was created.

Finally, it is important to mention that besides the common Python libraries, the tool also utilized two more custom ones: the `proxmox_automation.py`, which was used to communicate with the Proxmox API and the `utilities_automation.py`. The second library was used to perform smaller actions such as:

- Execute Terraform and Ansible code
- Interact with the `malware_list.txt` and the Ansible inventory

- Log information that was used for debugging and troubleshooting
- Convert JSON files to CSV, which is the main file type imported to the SQLite



4.3: Sandbox Service Flowchart

4.2.2 Deploying Windows Sandboxes

The first action was creating the Windows Virtual machine, which was made using the Infrastructure as Code (IaC) technology Terraform. In particular, the Terraform template was specifying the Virtual Machine's configuration, including 4 GB RAM size, 100 GB disk size, 2 CPU cores, and a network adapter for the isolated network.

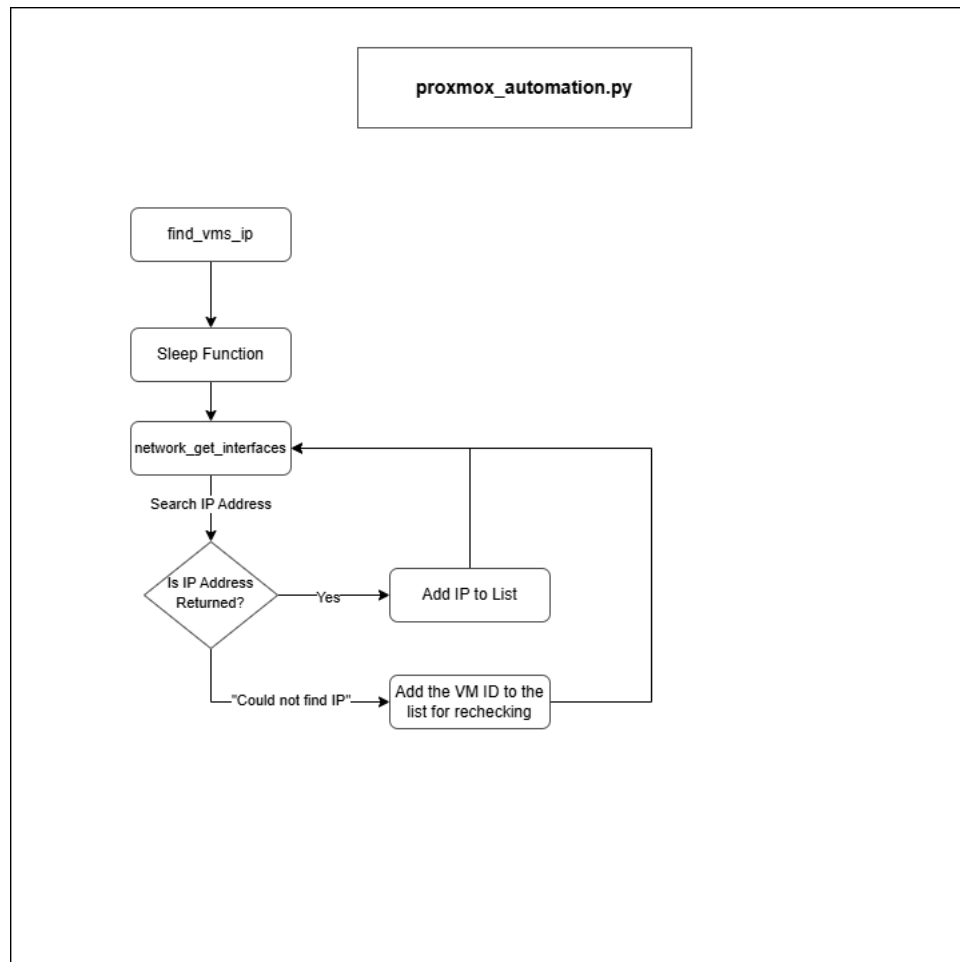
These specifications were selected since they are the bare minimum for Windows to function correctly. This was done to allow the creation of two more VMs by copying the code twice. Depending on the computing power of the Hypervisor, this feature could allow the creation of more Windows hosts simultaneously, thus making the analysis even faster.

The specified network adapter was also added to the PfSense firewall, which allowed traffic only toward the Internet and deny any traffic with destination the analysis network. This was done to isolate the network and protect other devices, such as the Hypervisor and the Ubuntu orchestrator, while also allowing the malware to communicate with malicious servers.

Finally, some additional important details from the template were that a Proxmox Agent needed to be installed since it was later used to identify the IPs of the hosts. Therefore a Windows Virtual Machine Clone ("Windows10-SandBox-Template") was created on the Hypervisor already containing the Proxmox Agent. The clone also had WINRM or SSH pre-configured to be accessed later.

4.2.3 Preparing Information for Ansible

After the Virtual Machines were created, Ansible accessed and configured them to prepare them for the malware's execution. A prerequisite for Ansible to work correctly was to provide it with the IPs of the hosts. This was achieved by creating a custom script that queried the Proxmox API for the IP address that corresponded to each interface of the virtual machine using the Proxmox Agent 4.4:



4.4: proxmox_automation Flowchart

After receiving the VM ID, the script slept for 30 seconds and then called the `network_get_interfaces` function to search for the network interfaces and their IPs. If the `network_get_interfaces` did not return an IP, the VM ID was added to the list of IDs that would be checked, and then the same loop was performed again. Finally, if the IP address returned the error `Could not find the IP of the VMID`, the ID was ignored and not added to the list.

The sleep functions were important since there was a delay between the completion of the Terraform actions, the creation of the VMs, and the execution of the Proxmox Agent.

4.2.4 Ansible Playbooks for Configuring the Sandbox

The first three Ansible Playbooks that were executed are `sandbox_setup.yaml`, `chocolatey_install.yaml`, and `download_sample.yaml`.

sandbox_setup.yaml

The Ansible Playbook namely “`sandbox_setup.yaml`” started by copying some necessary tools from the Ubuntu Orchestrator to the Windows Sandbox, more specifically, the tools:

- Sysmon: a Windows system service that enhanced logging.
- 7z: a file archiver used to uncompress the malware when installed and compressed the artifacts before collection.
- Wget: an open-source tool that retrieved malware samples from the Malware Bazaar website.

After copying the required binaries, the Sysmon service was created using the configuration file of SwiftOnSecurity. This is the most common configuration file, providing in-depth visibility on the host with logs such as Process Creation and Network Connection.

Finally, the last step was to add the entire file system, the C drive, to Microsoft’s Defender Antivirus Exclusion list. This was an important step when analyzing malware since it was desirable to allow the sample to perform as much activity as possible to identify its functionalities. If Defender was scanning the file system, it could stop the malware prematurely, thus losing important artifacts that could have been analyzed.

chocolatey_install.yaml

The second part regarding the configuration of the Virtual Machines was the chocolatey installation, a software management automation for Windows. This allowed to install multiple tools, such as Wireshark, without using the graphical interface or human interaction.

The first section of the Ansible Playbook executed a PowerShell command that downloaded and executed an installation script provided by chocolatey.org.

After that, the package manager was used to download both the Nmap and Wireshark tools. Nmap is a network scanning tool used to identify and enumerate hosts in a network. Although the tool was not used during this research, its installation also downloaded the Npcap driver used to capture network data. Currently, although the driver is free, its installation requires human interaction thus, the installation of the Nmap tool can help overcome this issue.

Wireshark, combined with the Npcap driver, allowed packet capturing on the host, significantly increasing the visibility of the malware’s behavior. Due to storage limitations, collecting endpoint artifacts from all their assets may be impossible for many organizations. Therefore, it is important to analyze samples and create security measures for the endpoint and the network artifacts.

download_sample.yaml

The final playbook utilized during this phase is the "download_sample.yaml." This playbook was executed for each host separately so a different sample could be downloaded each time.

When ran, Ansible first set the ExecutionPolicy to Unrestricted to allow the execution of PowerShell scripts by every process, but most importantly, Ansible. The second part of the playbook used the previously installed tool wget to communicate with the site Malware Bazaar and install new samples. This was done by performing a POST request towards the following API and providing the query="get_file&sha256_hash={file hash}:"

- <https://mb-api.abuse.ch/api/v1/>

Each file hash was obtained from the text file malware_list.txt, which will be later explained. The benefit of using Malware Bazaar was that new samples that still had their infrastructure functioning can be downloaded.

In the final step, Ansible unzipped the malware sample installed from Malware Bazaar using the password "infected". Compressing a sample with a known password is a common technique that security experts and researchers use to transfer them safely. It is also worth mentioning that the decompression was performed by the tool 7z, which was copied by the sandbox_setup playbook.

4.2.5 Execution of Malware Samples

During the execution phase, only the playbook `execute_sample.yaml` was ran, which was responsible for executing the `sysmonParser.ps1` PowerShell script with the necessary information.

More specifically, the playbook got all the executable files from the folder “Files” and stored them in a variable. The only executable in this folder was the malware, which is named differently every time. More specifically, Malware Bazaar will name its samples using the SHA256 hash corresponding to them. Therefore, the aforementioned approach was selected to keep this playbook flexible.

Now that the filename equal to the SHA256 hash, it can be provided to the PowerShell script as a parameter. `SysmonParser.ps1`’s purpose was to orchestrate the malware’s execution, the tools that need to run in parallel with it, and, most importantly, to parse the Sysmon logs correctly.

The first action it took was to use TShark, the command line version of Wireshark, to capture all the network traffic generated during the execution of the malware. After that, it executed the malware, waited for three minutes, and stopped the packet capture. Although malware could perform activities for more than three minutes or stay inactive for a significant period to avoid detection, the aforementioned timeframe was selected. This was the case since most malware samples, especially the ones used as Malware as a Service, perform automated activities when executed, create C2 communication, and then wait for the threat actor to provide commands.

When the script has done the above activity, it parsed the Windows Sysmon logs. Only the logs Process creation (Event ID 1) and Network connection (Event ID 3) were collected and analyzed during this research. Each process running in Windows is assigned a unique decimal number called process ID (PID) thus, the script searched for logs related only to the PID of the malware.

More specifically, it collected the process ID of the processes with their name equal to the file hash, which was how Malware Bazaar names the malware. After searching all the process creation logs for the process ID, it queried for logs with the parent process ID equal to the malware’s process ID. This was done in order to identify processes created by the malware. Finally, it searched the process creation logs again for the new process IDs and repeated this procedure until all the child processes were found.

Now that all process IDs related to the malware were collected, it will also queried the Network connection logs for them. Even though the process creation logs provided important information such as the file hash, file path, and command line, it was also important to investigate the malware’s network activity. Therefore the network logs were collected for all the malicious process IDs, which contained the file name and the IP address that the process communicated. These logs were also used to clarify which of the already collected packets were related to the malware.

4.2.6 Artifact Collection

The last part of this workflow was to collect the artifacts from the sandboxes and send them to the Ubuntu Orchestrator virtual machine. Due to the limitations of Ansible regarding the amount of data that can be copied, the packet capture size has to be reduced. Therefore, the packet captures have to be cleaned, and more specifically, packets not related to the malware were removed using the `remove_pcap_noise.yaml` playbook.

To remove the unwanted noise, the playbook first read a .csv file previously created by `Sysmonparser.ps1` while analyzing the Network connections logs. From the CSV file, only the Destination IP Addresses that the malware communicated with were queried. By providing this information to Tshark, only packets related to the aforementioned IP were kept, and the other unwanted noise was removed. After the new PCAP file were created, there were no need for the old one thus, it can be removed.

After all the data was collected, the final playbook `collect_artifacts.yaml` was executed, and the data was gathered after it was packaged. The folder Files containing all the artifacts was zipped using the tool 7z with the important detail that the output contained the name of the malware or, in other words, the SHA256 hash. This was an important detail since it ensured that when the artifacts from different sandboxes were being simultaneously stored on the Ubuntu Orchestrator, no files with the same name exist. If the files had the same name, there would be an overwrite of artifacts, and thus, important information would be lost.

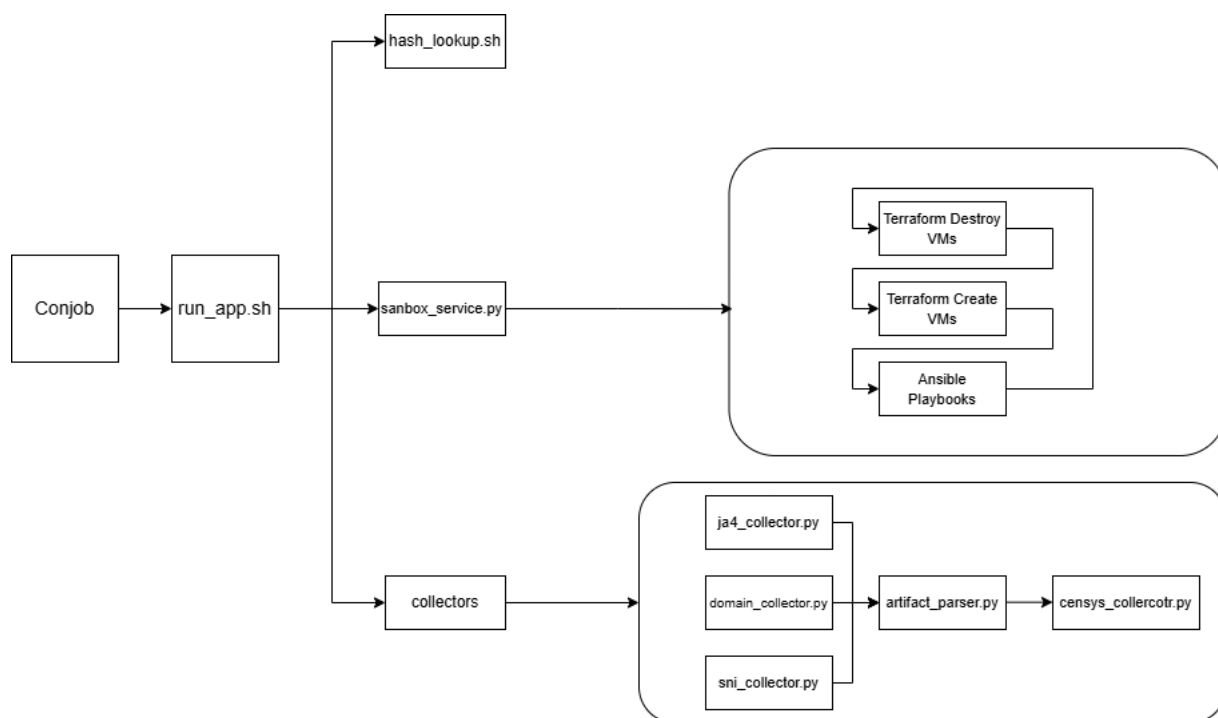
Finally, using the Ansible function `ansible.builtin.fetch`, the zip files were copied from the Sandboxes to the Ubuntu Virtual Machine. If the zip file do not have the malware's file hash as its name, then the analysis was unsuccessful, and an error message was printed as an output for troubleshooting reasons.

4.2.7 Collectors and Data Storage

The final step before the data analysis was storing the artifacts in the SQLite database after extracting some additional information. More specifically, the following artifacts were created:

- JA4+ Fingerprints: The JA4, JA4S, and JAX fingerprints were collected to signature the malware's network communication.
- Domains: To help analyze many samples, the URLs visited by the samples using unencrypted HTTP will be collected.
- Server Name Indications (SNI): SNI is an extension to TLS, which will be used to obtain the domains visited by the malware samples when the communication is encrypted.
- External Host Information: Using Censys, information was collected regarding the external IPs that malware samples communicated.

In order to achieve all of the aforementioned intelligence collection and the proper execution of the `sandbox_service.py` daily, a wrapper script was created. More specifically, it had the purpose of collecting the SHA256 hashes that were required by the `sandbox_service.py`, executing it, and when the process was finished, it started the collectors. The collectors were custom scripts interacting with the collected artifacts to extract the intelligence mentioned above. The graph 4.5 showcases the flow of the entire program:



4.5: Program Flow

It is also worth mentioning the `run_app.sh`, consisted of a for loop for each malware family that would be examined. This research analyzed AgentTesla, RedLineStealer, and RemcosRAT families since they were three of the most used malware.

After the loop started, the collector `hash_lookup.sh` was executed. This bash script requested all the samples of the selected malware family from Malware Bazaar. On top of that, when results were sent back in a JSON format, the tool `jq` was used to filter for samples uploaded "yesterday" and have the "exe" tag. The tool was used to analyze samples daily, and thus, there was no need to re-analyze older samples. It is also worth mentioning that the tool could execute only executable files (exe). However, creating an Ansible executor for a different file type would be effortless if it was necessary.

In the end, all the results were stored in `malware_list.txt`, which, as it was mentioned before, was read by `sandbox_service.py` to obtain SHA256 hashes that have to be analyzed.

Now that the hashes were collected, the `sandbox_service.py` was executed with the malware family as its argument. Afterward, the extraction of the JA4+ fingerprints, the HTTP URLs, and the TLS SNIs started.

ja4_collector.py

The first function being executed was " `find_collectable_artifacts`," the file `artifact_parser.py`, which contained a set of utility functions. The aforementioned function searched for all the files inside the folder Artifacts. In this research, those files were folders with the names of the Malware Families:

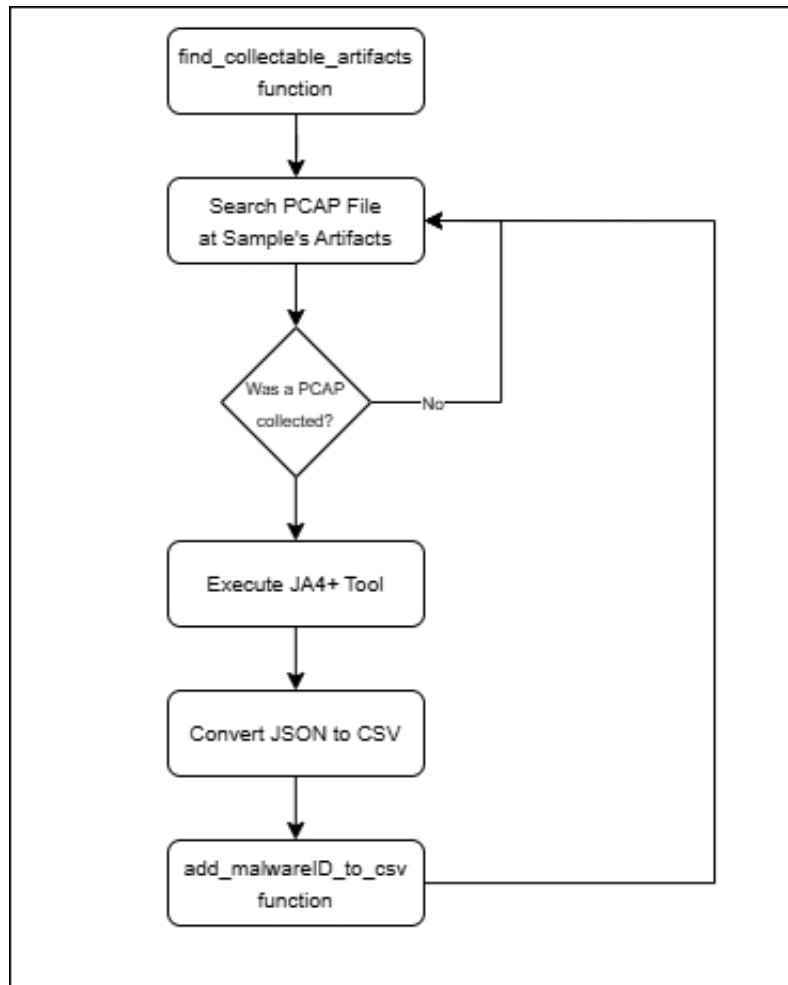
After that, it searched for any zip files inside them, tried to unzip them, and finally returned a list of the folders it managed to unzip. Those zip files contained the collected artifacts from each Windows Sandbox.

Now that all the folders were returned, the `ja4_collector.py` iterated through them and searched for .pcap files. In the case that it found one, it executed the tool `ja4.py` using the flag `-J` to make the output JSON and the flag `-f` to export the results into a file named `ja4.json`:

- `ja4.py malware-traffic.pcap -J -f ja4.json`

The final two functions that were executed were `create_csv_file_from_json` and `add_malwareID_to_csv`, which were also part of the custom `utilities_automation.py` file.

As explained, the `create_csv_file_from_json` converted JSON files to CSV. On the other hand, the `add_malwareID_to_csv` function added a new column containing the SHA256 hash to the `ja4.csv`. This was used as the malware ID, and its purpose was to help separate the samples during the analysis phase.



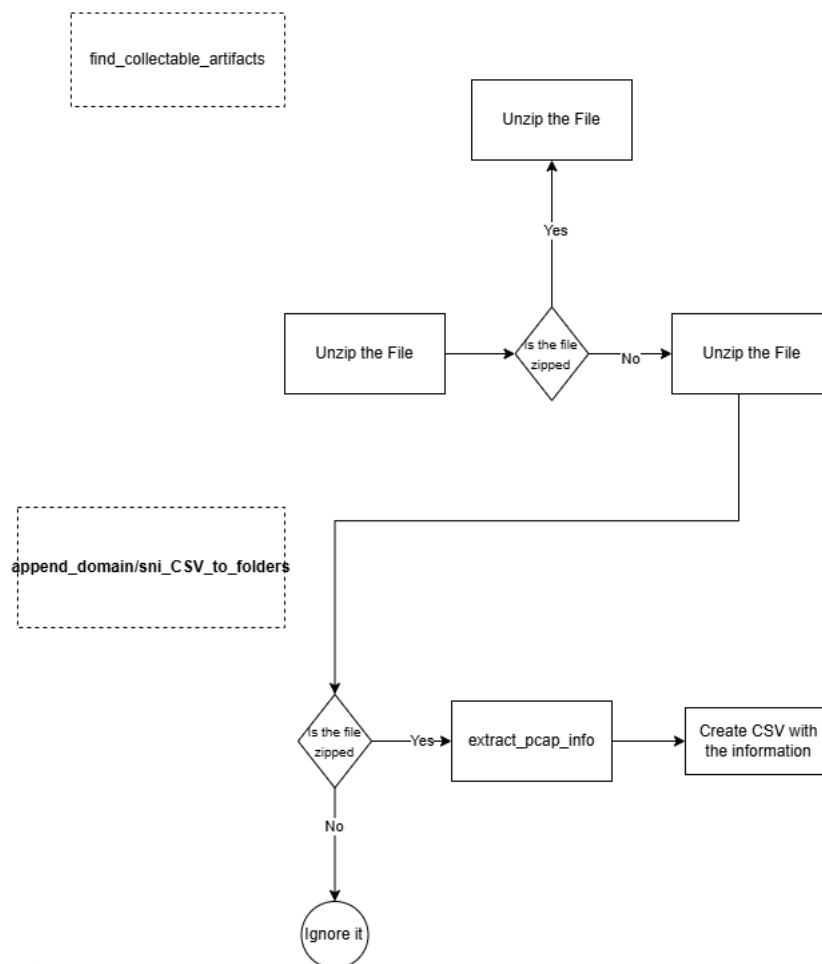
4.6: JA4 Collector Flowchart

domain_collector.py and sni_collector.py

The domain and SNI collector had the same logic as the JA4+ collector, but instead of using the `ja4.py` tool, it used TShark, the command line version of Wireshark, to query the packets for the URLs. More specifically, they executed the `find_collectable_artifacts` function. After that, they searched for `.pcap` files, and in the case that one is identified, the function `extract_pcap_info` was called, which executes the following command:

- HTTP: `tshark -r malwar_traffic.pcap -Y http.request -T fields -e http.host -e http.request.URI`
- TLS: `tshark -r malicious_traffic.pcap -Y tls.handshake.type == 1 -T fields -e tls.handshake.extensions_server_name`

In doing so, all the HTTP domains, HTTP URLs, and, in the case of TLS, the TLS SNIs were extracted from the PCAP files. Finally, similar to the ja4 collector, the results were saved as a CSV file with the name `domain.csv` and `sni.csv`.



4.7: Domain/SNI Collector Flowchart

artifact_parser.py

The next step in the workflow of the `run_app.sh` was the `artifact_parser.py`. Its primary assignment was to parse the CSV files and add them to the SQLite database in which they were stored. The first function used was `find_collectable_artifacts`, which found the paths of the artifacts. Afterward, it created the `id.csv`, which contained the SHA256 file hash and the Malware Family. It was used as an identifier to distinguish the malware and help with the analysis.

The next action the `artifact_parser.py` performed was to import the CSV files to SQLite using `append_csv_artifacts_to_sqlite`. Before storing the files, a new column named TimeStamp was created, which contained the current date. Thus, if desired, a historical analysis of the collected artifacts could be performed.

The function `search_files_in_folder` may be used redundantly. This design choice made the functions autonomous and not dependent on previous inputs, such as file location.

censys_collector.py

`Censys_collector.py` was the last collector used in this research. In contrast with the other collectors, this one was not used any collected artifacts but enriched the collected data. It queried SQLite for the destination IPs from the Network connection logs and searched them in Censys. To look up only the new hosts, the IP addresses from the Network connections and Censys intelligence databases were compared, and only the new ones were enriched.

After the IP addresses were selected, the script performed a request to the Censys API using the following URL

- `https://search.censys.io/api/v2/hosts/+ ip_address`

If the request succeeded, it started by parsing general information regarding an external IP, specifically the used Services, ports, DNS Records, Country, and Autonomous System. After that, by moderating through each port, the script collected the Extended Service Name and performed a more in-depth parsing of the used protocol.

Protocol-specific patterns were created for the HTTP, SSL, and SSH protocols since they were commonly found on C2 servers and could provide helpful information that can assist in fingerprinting a host. More specifically, information such as the Response Code, Bode Hash, and Title were collected from the HTTP protocol. The Certificate Issuer, Certificate Subject, and the JARM fingerprint were collected for the SSL protocol. Finally, the SHA256 fingerprint for the server key was gathered for the SSH protocol. Ultimately, this information was stored in the Censys Intelligence database for further analysis.

Chapter 5

Findings

5.1 Data Cleaning

Before the data analysis can begin, the non-functional samples must be identified and excluded from the operation. To achieve that, queries were performed in the SQLite database to identify patterns that indicate dysfunctional samples.

5.1.1 Defective Samples

During this research, endpoint artifacts were collected in the form of Sysmon logs, specifically the Process creation logs. On the other hand, network artifacts were in the form of Sysmon Network connection logs and PCAPs. Based on this visibility, samples that did not create any Process Creation or Network Connection logs can be considered defective since they did not provide any intelligence in this research. It is worth mentioning that a sample that did not create Network connection logs will not create any PCAP files. Therefore, the cleanup was performed only on the Network connection logs and not using the PCAPs.

Endpoint Activity

Starting with the endpoint logs, malware samples that did not spawn any child processes can be considered non-functional as far as endpoint visibility is concerned. To investigate this, an SQLite query searched the Malware Process Tree database for MalwareHashIDs that had been identified more than once. As explained, MalwareHashID was added to every database row and contains the sample's SHA256. Therefore, if a new process was spawned, it could be correlated to the original sample using this column.

When the aforementioned query was performed, 29 samples did not spawn a child process out of the 319 analyzed. Based on that, only a small number of samples, more specifically, 9%, were returned from this query. However, before drawing any conclusions, a review of the logs revealed another parameter that needs to be considered.

Werfault

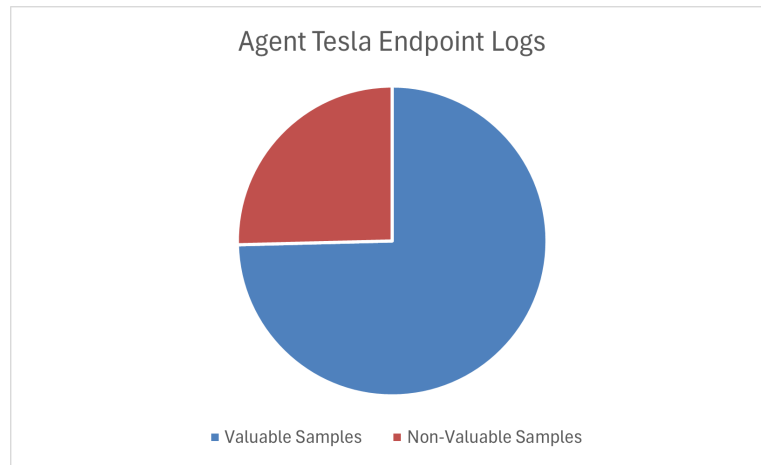
WerFault.exe is the Windows Error Reporting executable, this process spawns as soon as a process crashes. After that, it collects information and sends it to Microsoft Cloud. Based on

that further filtering should be made in order to exclude process trees that contain only the malware sample and werfault[31].

Firstly, in order to take into consideration the scenario that a malware sample spawned multiple child processes, only one of them crashed, and the other performed malicious behavior, some examinations had to be made. An example scenario is that:

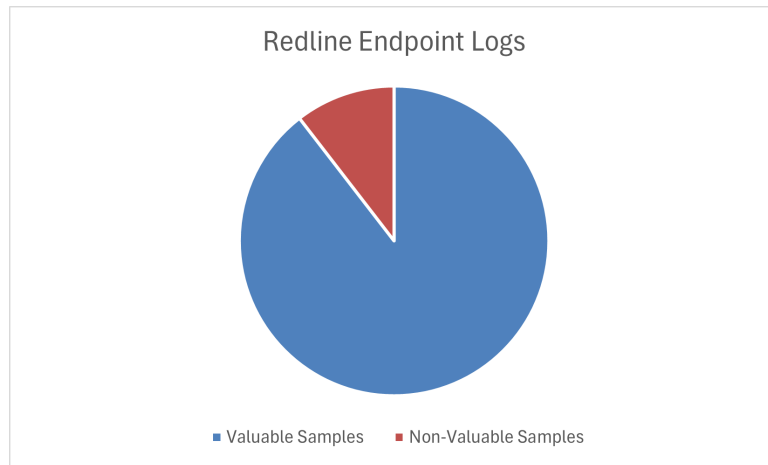
- `malware.exe` spawned `malware1.exe` and `malware2.exe`.
- `malware1.exe` crashed and spawned `werfault.exe`.
- `malware2.exe` performed C2 communication and allowed remote code execution to the malicious actor.

After analyzing all of the process trees that contain werfault, it could be concluded that no further processes were spawned. Therefore it is safe to proceed with the exclusion of process trees that contain werfault.



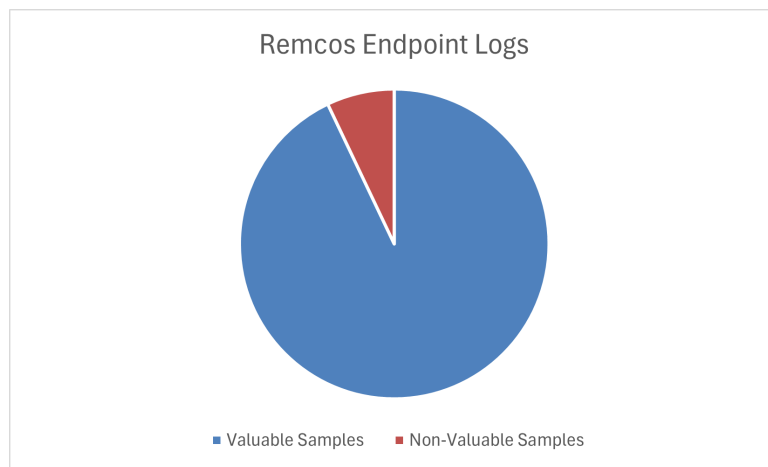
5.1: Non-valuable Agent Tesla Samples from Endpoint logs

As observed on the figure 5.1, the non-valuable samples had drastically increased, more specifically, by adding the werfault parameter, the dysfunctional samples were increased from 29 to 81. Therefore, the futile malware increased to 25% from 9%. This significant difference would dramatically change the research results if ignored.



5.2: Non-valuable Redline Samples from Endpoint logs

Conversely, adding the werfault to the default exclusion only resulted in a slight increase in the unneeded samples observed in the Redline dataset. More specifically, the dataset contained 277 samples, and the filtered-out ones were increased from 20 to 29, or from 7% to 10% of the samples.



5.3: Non-valuable Remcos Samples from Endpoint logs

Finally, during the cleaning of the Remcos dataset, from the 198 analyzed samples, 12 samples did not spawn a process, and 14 samples did not spawn a process spawned or spawned werfault, thus making the non-valuable samples 7% of the dataset.

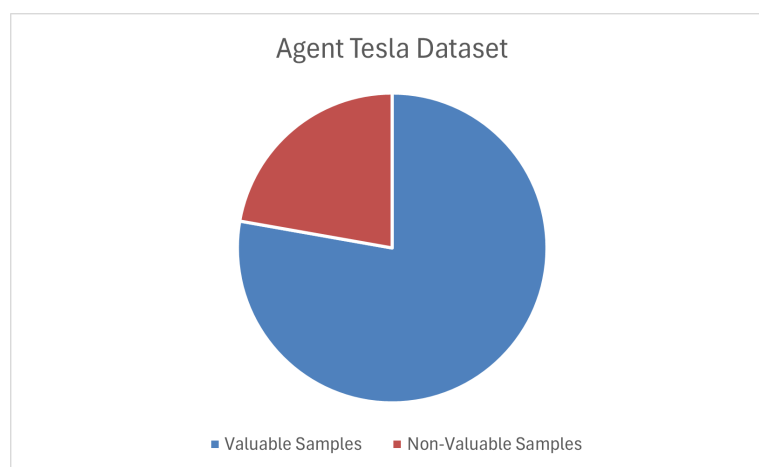
Based on the aforementioned results, Agent Tesla was the malware family with the most discarded samples. To summarize, the results showcased a significantly higher number of ignored malware in comparison with the other two families which had a similar percentage:

- Agent Tesla: 25%
- Redline: 10%
- Remcos: 7%

This phenomenon was observed because the Agent Tesla malware has a complex multi-stage architecture. More specifically, the malware can be delivered through various techniques, such as phishing emails or loaded directly to memory by loaders. This architecture also makes the malware's extraction from the loaders harder and may result in dysfunctional samples. Conversely, the previous observation may have occurred because the samples did not spawn child processes but still engaged in malicious activity.

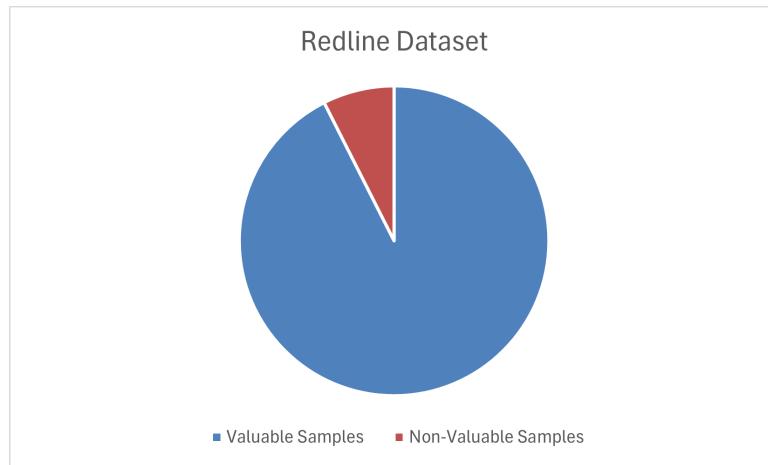
Network Activity

When investigating network-related activity, any sample's process tree that did not perform a network connection can be considered futile, and the malware can be discarded. To do so, a query was created to compare the samples in the Network Connection database with the Malware Profile Table, in which all the analyzed samples were stored. By combining the exclusions from the endpoint and network artifacts, the complete list of samples that could be ignored was obtained.



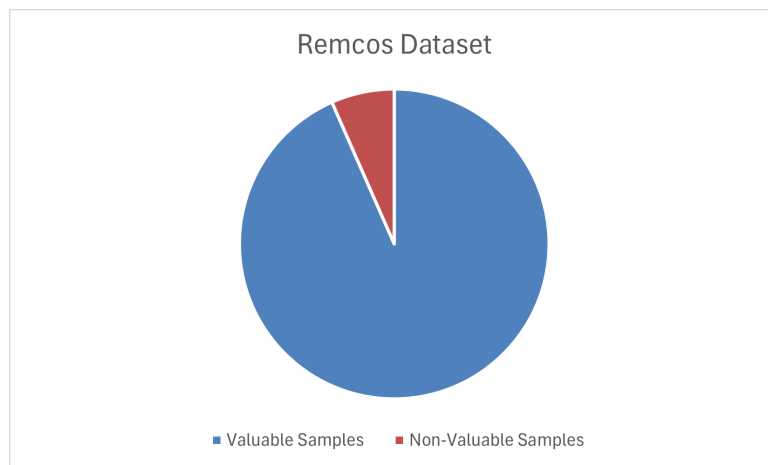
5.4: Non-valuable Agent Tesla from the dataset

Based on the aforementioned results, it was observed that some of the Agent Tesla that did not perform any endpoint activity performed a network connection. More specifically, from the original fruitless samples, 16% perform a network activity, and thus, the filtered-out samples were 68 and only 21% of the dataset, which is a significant decrease.



5.5: Non-valuable Redline from the dataset

On the other hand, when observing the Redline dataset, only 9 samples did not spawn a process that performed a network request. This provides an important distinction to the Agent Tesla family, which spawned fewer child processes but, regardless, performed a network connection.



5.6: Non-valuable Remcos from the dataset

Finally, in the Remcos dataset, the filtered-out samples decreased only by two, specifically from 14 to 12, thus excluding 6% of the traffic.

In summary, minor changes were made to the amount of the filtered-out samples in the Redline and Remcos datasets, which can be expected since they already had a small amount.

In contrast, a significant decrease in the excluded samples was observed in the Agent Tesla family. This insight indicates that Agent Tesla performs network connections by creating as little endpoint telemetry as possible.

5.2 Malware Insights

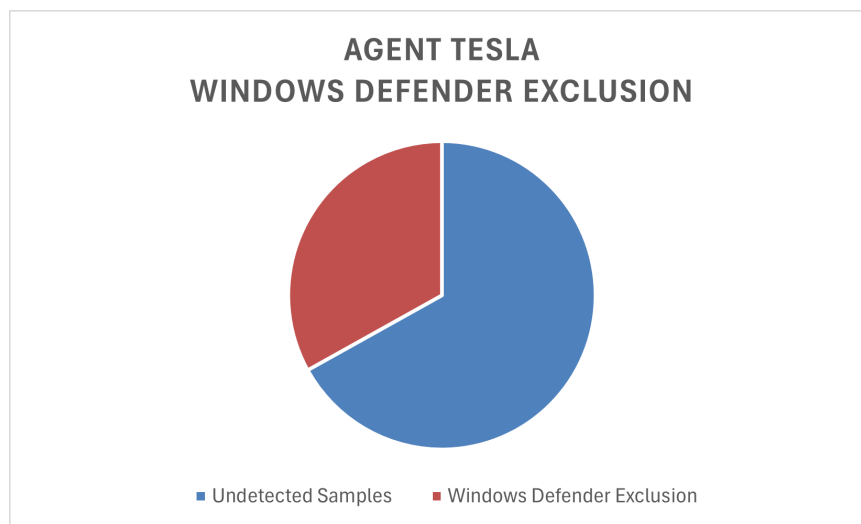
The analysis of the malicious samples was split into three main categories: the Process Tree, which contained endpoint artifacts, the Network Connections, which contained both network logs and HTTP traffic. Finally, the JA4 fingerprints in which the analysis of the TLS fingerprints as well as the certificate fingerprint was performed.

5.2.1 Agent Tesla Insights

Endpoint Analysis

During the endpoint telemetry analyses, it was identified that Agent Tesla samples were tampering with the Windows Defender Antivirus to avoid detection. More specifically, during their execution, they spawn PowerShell processes using the command "Add-MpPreference -ExclusionPath" and target themselves and copies of the malware in the App Data directory or the entire home directory. This was done in order to force the defender to ignore the malicious program and thus execute successfully. This technique is mapped on the MITRE framework with the title Impair Defenses: Disable or Modify Tools and the ID T1562.001[15].

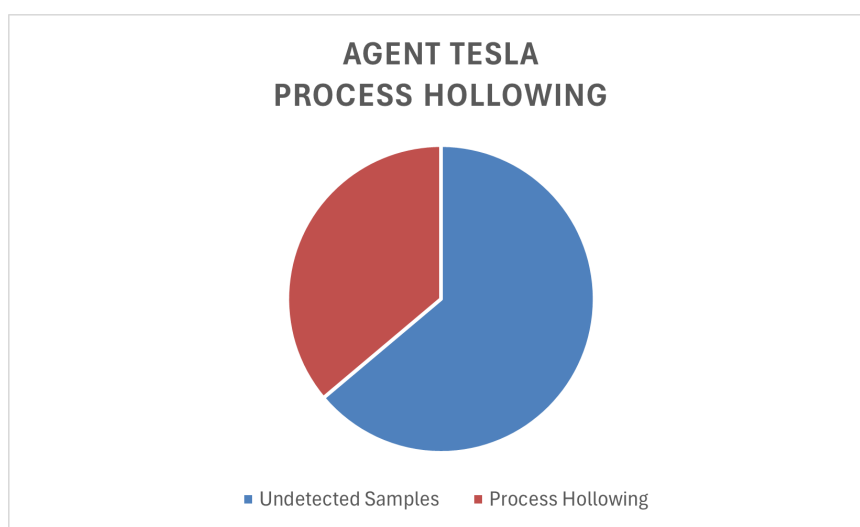
Based on the analysis, it was identified that 83 of the 249 functional samples tried to add an exclusion to Windows Defender using the command `Add-MpPreference -ExclusionPath`. The aforementioned behavior was critical in identifying the malware since it is used by 33% of all the samples.



5.7: Agent Tesla - Windows Defender Exclusion

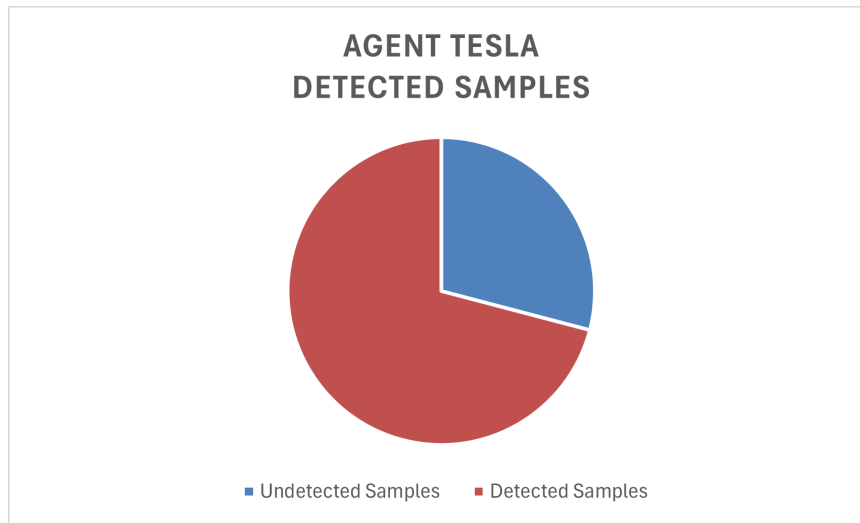
Observing the common child process created by Agent Tesla samples, suspicious behavior from legitimate binaries could be identified. More specifically, network connections were performed from legitimate Microsoft processes. The artifacts collected in this research did not clearly indicate what this procedure is, but multiple research articles on the malware confirm that it was Process Hollowing. The fact that no command-line arguments were used during process execution also proves this theory since it was the most common method to perform it.[16]

Process Hollowing is a process injection technique during which the malware spawns a legitimate process, suspends it, and injects malicious code into its memory. Due to the usage of legitimate and signed Microsoft processes, the malware may be able to evade process-based defenses. This technique is mapped on the MITRE framework with the title Process Injection: Process Hollowing and the ID T1055.012[17]. The above technique was used by 95 samples, or 38% of the functional malware, making it an important behavior pattern.



5.8: Agent Tesla - Process Hollowing

By combining the two behaviors and comparing them with the dataset, it could be concluded that 178 samples, or 71% of the dataset, can be detected. Although a significant amount of malware samples were detected, the detections can be increased by taking advantage of the Network traffic and the JA4 Fingerprints.



5.9: Agent Tesla - Detected Samples

Network Analysis

Due to the dynamic analysis of the malicious samples and the subsequent collection of the Network connection logs, 83 malicious IP addresses could be collected.

Based on the investigation, it was concluded that the Agent Tesla family uses the ports 80 (HTTP) and 443 (HTTPS) for C2 communication, which are widespread protocols, and thus, the traffic could be stealthier. It is also worth mentioning that ports 587 and 25 were used to exfiltrate sensitive data via email. Finally the port 21 and the protocol FTP was also used for exfiltration.

The Simple Mail Transfer Protocol (SMTP) analysis concluded that the malware used both encrypted and unencrypted versions. Since the encrypted version can be better analyzed in the JA4 section, this one will focus on the unencrypted traffic. When inspecting the unencrypted traffic, it was identified that after the SMTP protocol was established, the infected host exfiltrated host information, collected data, and password using an email. More specifically, the malware always forwarded the current timestamp as Time, the User Name, the Computer Name, the OS Full Name, the CPU, and the RAM.[25] Except for the above, it could also exfiltrate credentials from various applications, such as Google Chrome and Firefox, using the pattern:

- `<hr> Host:
 Username:
 Password
 Application <hr>`

Finally, another important observation was that the mail subject contains the username and the hostname and has the following naming convention:

- Mail Subject: `PW_<username>/<hostname>`

Similarly with the unencrypted SMTP method, the malware developers had also implemented a different approach in the form of exfiltrating data using FTP. File Transfer Protocol (FTP) is a protocol built on top of TCP/IP that is used to transfer files between hosts. In the specific example, the attacker sent a file with the naming convention[26]:

- PW_<username>-<hostname>_<date>.html

The aforementioned file had a similar structure to the email used before and contained, once again, the exfiltrated data 5.10:

↑ Send: 48 b	Timeshift: 46273 ms	Download	Hide
00000000	53 54 4F 52 20 50 57 5F 61 64 6D 69 6E 2D 55 53	STOR PW_admin-US	
00000010	45 52 2D 50 43 5F 32 30 32 34 5F 30 33 5F 30 37	ER-PC_2024_03_07	
00000020	5F 31 39 5F 30 30 5F 33 36 2E 68 74 6D 6C 0D 0A	_19_00_36.html..	

5.10: Agent Telsa FTP C2 Communication

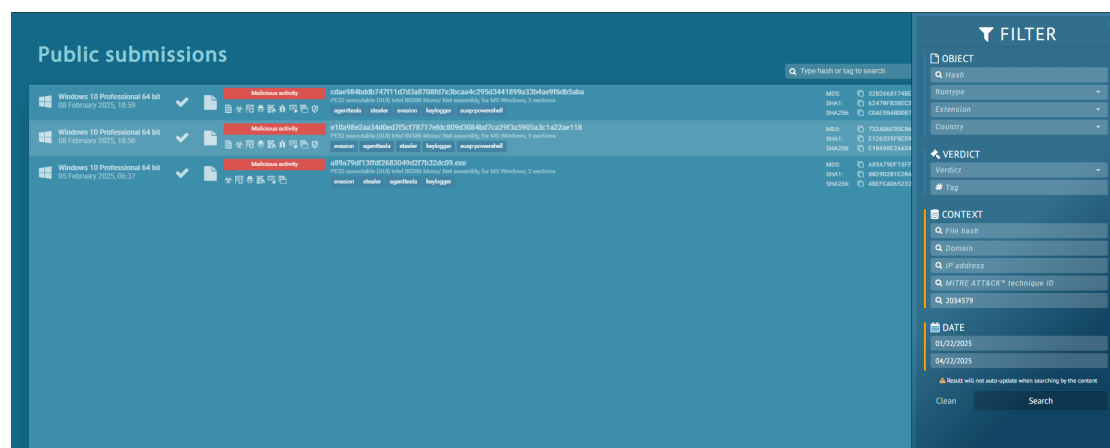
Based on multiple older research studies on the Agent Tesla malware, it was identified that it also uses unencrypted requests to perform C2 communication and exfiltrate data. More specifically, a POST request is typically performed to the domain containing the variable p, which is equal to a based64 encoded command[27].

↑ Send: 460 b	Timeshift: 22347 ms	⌵ Download	Hide
00000000	70 30 72 38 56 50 45 50 47 25 32 42 49 73 4F 35	p=r8VPEPG'2BIs05	
00000010	2F 5A 72 73 25 32 42 68 63 69 65 59 33 41 4B 46	/Zrs'2BhcieY3AKF	
00000020	77 49 7A 46 59 65 71 43 46 36 4A 55 42 62 35 35	wIzFYeqCF6JUBb55	
00000030	47 57 65 4C 32 52 66 7A 6F 69 6D 63 53 44 68 57	GWel2RfzoimcSDkW	
00000040	62 78 4C 52 4E 44 6E 39 74 62 76 49 4E 76 43 72	bxLRNdn9tbvINvCr	
00000050	4E 4E 34 4A 44 62 69 39 68 4C 56 63 35 6F 33 33	NN4JDbi9kLVc5o33	
00000060	51 78 47 62 51 57 56 4D 79 47 54 4E 66 34 6C 34	OxGbQWVMyGTNf414	
00000070	6A 68 4C 4A 6E 42 4D 79 68 36 45 44 78 54 48 6A	jklJnBMyk6EDxTHj	
00000080	79 71 4E 38 74 25 32 42 6F 31 71 6E 58 4F 39 38	yqN8t'2Bo1qnX090	
00000090	78 30 78 6F 36 2F 63 34 69 77 53 51 62 73 55 79	x0p06/c4iwSQbsUy	
000000a0	41 6D 65 6D 76 68 31 43 76 70 6E 6A 36 79 4D 59	Ameavk1Cvnpj6yMY	
000000b0	62 74 4E 7A 4C 44 51 5A 32 42 44 4E 61 63 56 63	btNzLDQZ2BDNacVc	
000000c0	68 68 42 25 32 42 77 59 38 50 61 61 4B 61 2F 47	khB'2BwY8PaaKa/G	
000000d0	66 4D 4E 69 78 71 74 31 37 56 37 34 54 75 68 68	fMNixqt17V74Tuhk	
000000e0	71 37 38 74 7A 72 25 32 42 67 4D 68 79 31 41 7A	q78tzr'2BgMhy1Az	
000000f0	73 36 67 5A 46 2F 25 32 42 30 58 50 37 43 59 54	s6gZF/'2B0XP7CYT	
00000100	48 55 58 6D 5A 52 45 6F 4C 45 79 74 42 48 79 79	HUXmZREoLEytBhyy	
00000110	68 51 55 55 71 41 33 54 68 31 31 67 39 6A 38 48	hQUUqA3Tk11q9i8H	

5.11: Agent Telsa HTTP C2 Communication

Despite that, the analysis and the investigation conducted in this research did not show Agent Tesla's sample performing C2 communication with HTTP. Similar results were observed after pivoting the investigation to the online Malware Analysis platform Anyrun and examining recent samples. This showcases that either the malware family has evolved and has started to use more uncommon exfiltration techniques or that the threat actors have abandoned HTTP communication for different protocols such as SMTP.

Anyrun is a cloud-based malware analysis service that performs advanced dynamic analysis of samples. This service executes Suricata rules on the network traffic generated by the malware and also allows searching samples that trigger a specific Suricata detection. Based on the analysis performed on the sample with the MD5 hash `ae4a2fc900ca8ee22a8045770f439e64`, the Suricata detection that corresponds to the AgentTesla HTTP C2 communication is called ET MALWARE AgentTesla Communicating with CnC Server and has the ID 2034579. As can be observed from the following screenshot, during the last three months of the time of the research, only four samples triggered this detection, indicating that this technique has been abandoned [5.12](#):



5.12: Anyrun for Agent Tesla

Another functionality of the Agent Tesla malware family that can be observed from the network telemetry was collecting the host's external IP using the domain `apify.org`. More specifically, when performing a request to `api.apify.org`, the domain responded with an HTTP response containing the host's external IP. By doing so, the malware sample could confirm that it had internet connectivity and also collect information that could later be exfiltrated.

The final technique identified for C2 communication and exfiltration was using a Telegram channel. Telegram is a messaging application that allows users to communicate. Since it is considered a legitimate application, malicious actors have created malware that can access users' chat, acquire commands, or collect and exfiltrate sensitive data. Based on the performed analysis, only five samples performed such communication, 1.5% of the samples indicate that the other methods of C2 communication were replacing this technique.

JA4+ Fingerprints

JA4, the TLS Client Fingerprint, is used to fingerprint an application using information from the SSL Client Hello packet. When investigating the JA4 fingerprints of the AgentTesla samples, it was concluded that only the `t12d210700_76e208dd3e22_2dae41c691ec` was used.

- Protocol: TCP
- TLS Version: 12
- SNI: Domain
- Number of Ciphers: 21
- Number of Extensions: 07
- First ALPN: No ALPN

It is important to note that the specific JA4 fingerprint is equal to the one generated by the underlying functions and C# library used by the malware. Thus, it has to be correlated with other information to provide greater value.

JA4S, the TLS Server, used to fingerprint applications using information from the SSL Server Hello packet. The server hello message is the server's response and contains the CipherSuite chosen by the server from the list provided by the client, the session ID, and another random byte string. Since the server hello packet chose information that the client hello packet contained, the JA4 and the JA4S fingerprints are relative to each other. Based on the analysis performed on the JA4S, no patterns could be identified, and thus, it was concluded that different Server applications or application versions were used.

During the investigation of the Certificates used to encrypt the TLS it was identified that most of the Issuers of the certificates were **Let's Encrypt**. Let's Encrypt is a Certificate Authority that provides free TLS certificates, allowing websites to enable HTTPS encryption making their connections secure. It is also worth mentioning that most of the values in the Subject Certificate were the domain. In other words, the domain `padinet[.]com` had the certificate subject `padinet[.]com`.

JA4X fingerprints, are the way in which TLS certificates are generated, not the values within the certificate. Threat actors will create different certificates but tend to use the same methods to create said certificates, thereby having the same JA4X fingerprint. During the analysis the most common combination of JA4X fingerprints was `a373a9f83c6b_7022c563de38_821a8ec155c6` and `a373a9f83c6b_a373a9f83c6b_0d8df11fc25a`.

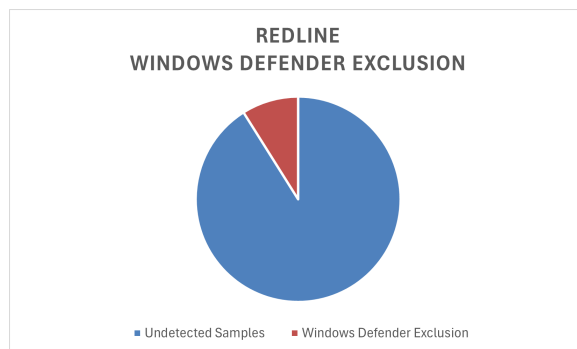
Fingerprint	Distinguished Name Fields
a373a9f83c6b	CN (Common Name)
7022c563de38	C (Country), ST (State or Province), L (Locality), O (Organization), CN (Common Name)

Table 5.1: Certificate Fingerprints and Corresponding Distinguished Name Fields

5.2.2 Redline Insights

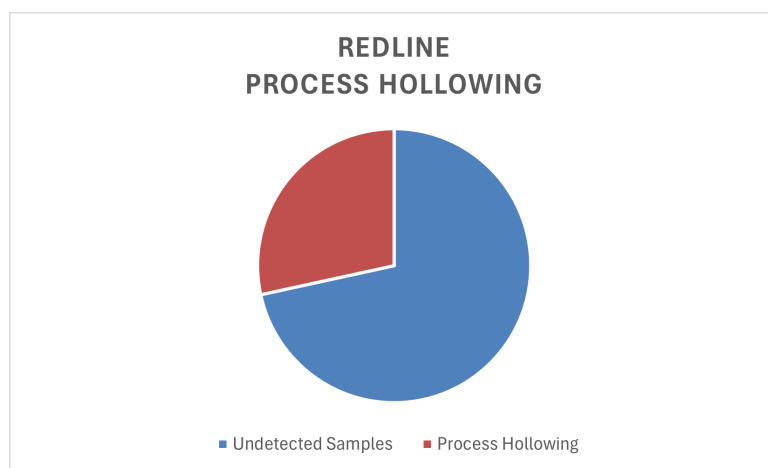
Endpoint Analysis

In conjunction with the Agent Tesla malware family, the Redline malware was identified using multiple techniques at the endpoint. Redline malware samples were detected to be interfering with Microsoft Windows Defender by excluding their locations from the scanned folders. The technique Impair Defenses: Disable or Modify Tools, with the ID 1562.001, was performed by 23 samples, 8% of the functional samples.



5.13: Redline - Windows Defender Exclusion

Another technique used by the Redline and the Agent Tesla malware families was Process Hollowing, which targeted legitimate files. This attack was used most commonly by the malware, and thus, 73 samples, or 28% of the functional samples, were detected performing such activities.



5.14: Redline - Process Hollowing

Furthermore, another Tactic, Technique, and Procedure (TTP) that was detected was the use of a one-character executable. More specifically, five samples (2% of the functional samples) were detected executing a file named 1.exe from the Temp directory. Although a low percentage of the samples observed this phenomenon, it is an important observation since it is heavily utilized by malicious actors, especially during the later phases of an infection.

Additionally, the MITRE technique T1036: Masquerading was identified through the creation and execution of .cmd or .bat files via cmd.exe one-liners. The cmd command moved a file to the same directory and changed its file extension, essentially renaming it. After that, the newly created file was executed, and the cmd prompt was closed using the command exit[35]. Only five samples or 2% of the functional dataset performed such commands.

CMD One-liners Examples
"C:\Windows\System32\cmd.exe" /c move Batch Batch.bat & Batch.bat
"C:\Windows\System32\cmd.exe" /c move Glucose Glucose.bat & Glucose.bat

Table 5.2: Windows Command Line One-Liners

Finally, the creation of scheduled tasks using the following command line was observed.

Schedule Task Creation
"C:\Windows\System32\cmd.exe" /C schtasks /create /tn MyApp /tr %%APPDATA%\service.exe /st 00:00 /du 9999:59 /sc daily /ri 1 /f

Table 5.3: Command to Create a Scheduled Task

As it can be recognized, the scheduled task ran an executable from the Appdata directory, had an abnormal duration, and was executed daily. It is important to note that the scheduled task creation was performed from only one sample and thus it cannot be considered as a common technique used by the malware during its execution.

In conclusion, by combining all of the aforementioned observations, 114 different samples could be observed, thus detecting 44% percent of the functional samples.

Network Analysis

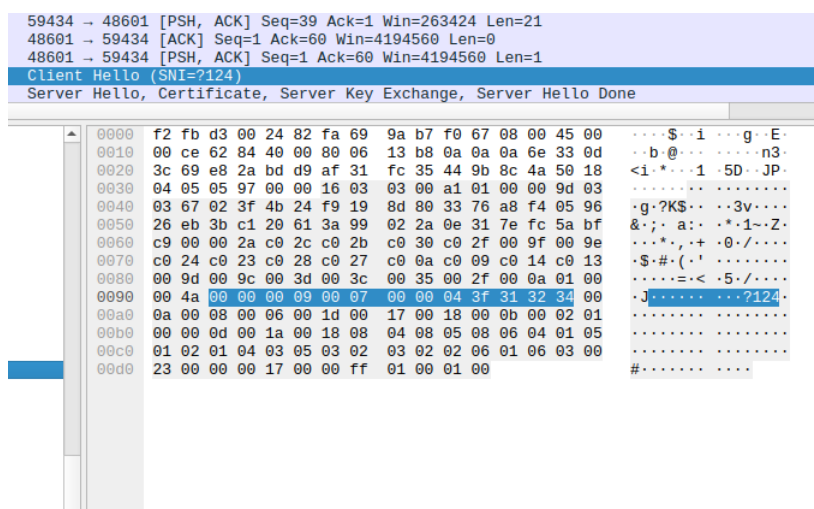
The most common destination ports identified from the analysis of the Redline malware were first the 443 and 80, which are related to the HTTP and HTTPS protocols. After that, connection to 37 different ports was observed, most of them being four to five digits long.

After further investigation, it was concluded that the communication towards the default HTTP and HTTPS ports was mostly towards api.ip[.]sb, a site used to obtain the external IP of the infected host. On top of that, the rest of the traffic was directed towards legitimate domains used for malicious purposes, such as Telegram and Steam communities. The analysis of the rest of the traffic showcased, the the high numbered ports were used as a C2 server for the Redline malware.

Another important remark was that beaconing was performed during the C2 communication. More specifically, when the Redline malware communicated with its malicious command and control server, many requests were performed instead of creating one long connection. This different approach could allow an attacker to read commands or exfiltrate data to the malicious server and simultaneously avoiding network detections for long duration sockets.

The Redline malware family used two main techniques for C2 communication. It either used encrypted TCP connections or unencrypted SOAP protocol. Simple Object Access Protocol (SOAP) is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks.

The encrypted Command and Control communication utilized TLS, thus during this analysis, not a lot of information could be acquired. Despite that, an important anomaly was pinpointed, more specifically, an SNI value is equal to ?124. Server Name Indication (SNI) is an extension of the HTTPS TLS protocol. The extension makes it possible to specify the website's domain name during the TLS handshake instead of when the HTTP connection opens after the handshake, making the communication more efficient.



5.15: Redine Suspicious SNI

The SOAPAction header is a transport protocol header transmitted with a SOAP message. It is used to route the request SOAP message. The Redline family heavily utilizes this header, and two variations were identified.

The first variant utilized the SOAPAction with one of the following values:

Option	Description
tempuri.org/Endpoint/CheckConnect	Verify if the connection is functioning properly
tempuri.org/Endpoint/EnvironmentSettings	Requests file to collect
tempuri.org/Endpoint/SetEnvironment	Uploads stolen data
tempuri.org/Endpoint/GetUpdates	Uploads stolen data and requests for additional tasks

Table 5.4: Redline C2 Options

```

HTTP/1.1 100 Continue

POST / HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/Endpoint/CheckConnect"
Host: 45.137.22.242:55615
Content-Length: 137
Expect: 100-continue
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version='1.0'><Envelope xmlns='http://schemas.xmlsoap.org/soap/envelope/'><s:Body><CheckConnect xmlns='http://tempuri.org/'></s:Body></Envelope>
HTTP/1.1 200 OK
Content-Length: 212
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Mon, 22 Jul 2024 10:38:13 GMT

<?xml version='1.0'><Envelope xmlns='http://schemas.xmlsoap.org/soap/envelope/'><s:Body><CheckConnectResponse xmlns='http://tempuri.org/'><CheckConnectResult>true</CheckConnectResult></CheckConnectResponse></s:Body></Envelope>HTTP/1.1 100 Continue

```

5.16: Redine First SOAP Variant[18, 19]

The second variant used the SOAP protocol and the SOAPAction header similarly, containing, once again, the value tempuri.org. In contrast with the previous variant, this variant contained 24 different ID parameters that perform different actions. The command and control used the convention http://tempuri.org/Entity/Id(1-24).

```

E1..UNKNOWNES3.....
..p&http://tempuri.org/Entity/Id23Response.Id23Response
Id23Result.Entity6.Current.Filter
FinalPoint.Status.VisibleV...s...a.V.D
.....D.....K...8...UID.....V.B..
.B....b....i.E..E....E....900000E..E#.-https://evoto-pc.ru/pr.exe%tmp%\filename.exeE%..2E).E....ActiveE.....

```

5.17: Redine Second SOAP Variant[24]

It is worth mentioning that the tempuri.org domain is owned by Microsoft and redirects to their Bing search engine. This domain is the test default namespace URI used by Microsoft development products.

Another network technique that the malware utilized was communication towards the URL api[.]ipify[.]org/ip. This domain is used to obtain the visitor's external IP, similar to the domain api[.]ipify[.]org used by the Agent Tesla family. Once again this technique was utilized by the malicious actor to obtain the information and to check for internet connectivity.

JA4+ Fingerprints

During the analysis of the JA4 TLS Client Fingerprint it was concluded that it has the same value `t12d210700_76e208dd3e22_2dae41c691ec` as the Agent Tesla JA4 fingerprint. With the above results and after confirming that they are both written in *C#*, it could be concluded that both of the malware families used similar libraries for communication with external hosts.

- Protocol: TCP
- TLS Version: 12
- SNI: Domain
- Number of Ciphers: 21
- Number of Extensions: 07
- First ALPN: No ALPN

When investigating the JA4S, the TLS Server Response fingerprint had five different values:

JA4S Fingerprint
<code>t120200_c030_5333cdffa7d9</code>
<code>t100500_c013_b508675693ed</code>
<code>t120500_c02f_45145cc9ca06</code>
<code>t120500_c030_45145cc9ca06</code>
<code>t120400_c02c_f050742559f7</code>

Table 5.5: Redline JA4S Fingerprints

Further analysis showed that the fingerprint `t100500_c013_b508675693ed` belonged to the legitimate domain `api[.]ip[.]sb`. The fingerprints `t120500_c02f_45145cc9ca06`, `t120500_c030_45145cc9ca06`, `t120400_c02c_f050742559f7` are related to HTTPS communication to the legitimate applications Telegram, Steam, and the malicious domain `evoto-pc[.]ru` which could not be analyzed since it was offline during the investigation. Finally, the JA4S fingerprint `t120200_c030_5333cdffa7d9` was utilized when communication occurred towards high destination ports, and it is associated with communication to the C2 server, making it the only fingerprint related with malicious infrastructure.

The Redline malware family did not have any certificates when communicating with encrypted traffic. More specifically, both the subject and the issuer certificate value were `id-at-commonName=localhost`.

9 0.219650 95.216.107.53 10.10.10.223 TLSv1.2 628 12311 59300	Server Hello, Certificate, Server Key Exchange
<ul style="list-style-type: none"> Frame 9: 628 bytes on wire (5024 bits), 628 bytes captured (5024 bits) on interface \Device\NPF_{17172B42-7A5A-44...} Ethernet II, Src: f2:fb:d3:00:24:82 (f2:fb:d3:00:24:82), Dst: 7a:33:62:09:5a:79 (7a:33:62:09:5a:79) Internet Protocol Version 4, Src: 95.216.107.53, Dst: 10.10.10.223 Transmission Control Protocol, Src Port: 12311, Dst Port: 59300, Seq: 2, Ack: 227, Len: 574 Transport Layer Security <ul style="list-style-type: none"> TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages <ul style="list-style-type: none"> Content Type: Handshake (22) Version: TLS 1.2 (0x0303) Length: 569 Handshake Protocol: Server Hello <ul style="list-style-type: none"> Handshake Type: Server Hello (2) Length: 81 Version: TLS 1.2 (0x0303) Random: 66e5447bf7cc177e9c910f70f64a1bb07de5ab82b00fdd53941239339cd36 Session ID Length: 32 Session ID: 45230000fc85f5f3be01ea589682fce8d3502f6cc7b00fc513b067ae2caabf51 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) Compression Method: null (0) Extensions Length: 9 Extension: extended_master_secret (len=0) Extension: renegotiation_info (len=1) <ul style="list-style-type: none"> [JA3S Fullstring: 771,49200,23-65281] [JA3S: ae4edc6faf64d08308082ad26be60767] Handshake Protocol: Certificate <ul style="list-style-type: none"> Handshake Type: Certificate (11) Length: 303 Certificates Length: 300 Certificates (300 bytes) <ul style="list-style-type: none"> Certificate Length: 297 Certificate [0] 663082201253081d0a003020102021022d6da97f75bf4814124267de0ca067d390d06092a864886f70d01010 signedCertificate <ul style="list-style-type: none"> version: v3 (2) serialNumber: 0x22d6da97f75bf4814124267de0ca067d signature (sha1WithRSAEncryption) issuer: rdnSequence (0) <ul style="list-style-type: none"> rdnSequence: 1 item (id-at-commonName=localhost) <ul style="list-style-type: none"> RDNSequence item: 1 item (id-at-commonName=localhost) <ul style="list-style-type: none"> RelativeDistinguishedName item (id-at-commonName=localhost) <ul style="list-style-type: none"> Object Id: 2.5.4.3 (id-at-commonName) DirectoryString: printableString (1) <ul style="list-style-type: none"> printableString: localhost validity <ul style="list-style-type: none"> subject: rdnSequence (0) subjectPublicKeyInfo algorithmIdentifier (sha1WithRSAEncryption) padding: 0 encrypted: 43772396b09c4ec05d10306d60c7c93fedc806e638bfd888c755062672144499944272f36f48b0d5f12643fed 	<pre> 0000 7a 33 62 09 5a 79 f2 fb d3 00 24 82 08 00 45 00 0010 02 66 e4 2e 40 00 75 06 3f 6d 5f d8 6b 35 0a 0a 0020 0a df 30 17 e7 a4 7a 23 92 58 7e e1 b1 c2 50 18 0030 20 02 0f 4f 00 00 16 03 03 02 39 02 00 00 51 03 0040 03 66 e5 44 7b 7c c1 77 e9 c9 10 f7 0f 64 a1 0050 bb 07 8d e5 ab 82 b0 00 fd d5 39 41 23 93 39 cd 0060 36 20 45 23 00 00 fc 85 f5 f3 be 01 ea 58 96 82 0070 fc e8 d3 50 2f 6c c7 b0 0f c5 13 b0 67 ae 2c aa 0080 bf 51 c0 30 00 00 09 17 00 00 ff 01 00 01 00 0090 0b 00 01 2f 00 01 2c 00 01 29 30 82 01 25 30 81 00a0 00 00 03 02 01 02 02 10 22 06 da 97 f7 5b f4 81 00b0 41 24 26 7d e9 ca 06 7d 30 0d 06 09 2a 05 48 05 00c0 f7 0d 01 01 05 05 00 30 14 31 12 30 10 00 03 55 00d0 04 03 13 09 06 6f 63 61 6c 60 6f 73 74 30 1e 17 00e0 0d 32 34 30 39 30 35 30 37 31 31 31 33 36 5a 17 0d 00f0 32 39 30 39 31 32 39 37 31 31 33 36 5a 30 14 31 0100 12 30 10 06 03 55 04 03 13 09 06 6f 63 61 6c 68 0110 6f 73 74 30 5c 30 0d 06 09 2a 86 48 86 f7 0d 01 0120 01 01 05 00 03 4b 00 30 48 02 41 00 c3 2b 2d b4 0130 6e 86 97 77 78 60 ac d8 7c 04 c3 d7 19 7c 0a d3 0140 01 5e b0 db 70 10 84 cf a9 8b d6 81 25 2d fc 46 0150 23 8c d2 c6 5f 6d 8a c3 ea 25 f0 7f 55 24 f8 16 0160 c7 0b 8f 09 1b 61 16 52 e8 a3 a1 11 02 03 01 00 0170 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00 0180 03 41 00 43 77 23 96 b0 9c 4e c0 5d 10 30 6d 60 0190 c7 c9 3f ed c8 06 e6 38 bf d8 88 c7 55 06 26 72 01a0 14 44 99 94 42 72 f3 6f 48 b0 05 f1 26 43 fe 03 01b0 3c 09 35 83 ca 03 e8 b3 c9 ba 00 db e6 42 4f df 01c0 a0 5d 3a 0c 00 00 a9 03 00 13 61 04 d9 59 b1 0f 01d0 cf 69 79 e1 40 11 c5 9b 50 48 ed 7a dc 2e 58 2b 01e0 fc 6c 8f 09 2e 97 33 a4 c6 66 b2 2d 88 c5 84 c5 01f0 1e 71 ac 2f 85 32 aa f6 fb b6 0b 8c ea c1 f8 f9 0200 1e 2c be 69 40 ef 2d 68 a1 fa 60 f9 b3 47 dc 0b 0210 f8 de 5c fb 4b 13 e9 f3 db 34 03 5b 07 0a c7 0b 0220 d2 6d b0 08 f2 8c 33 22 3b a6 2d 0a 04 01 00 40 0230 5c 19 2a 6c 39 64 a1 6f 51 3c 14 24 f7 db eb 17 0240 ea bb 82 dc 11 23 9c e3 a9 10 ed f0 66 bf e7 c6 0250 e2 a4 6a ee 5c 87 3d 5b b7 38 c9 2c cf 8c 14 ce 0260 75 27 99 18 bc 04 82 ce ca f7 6f 9b 65 56 11 fb 0270 0e 00 00 00 </pre>

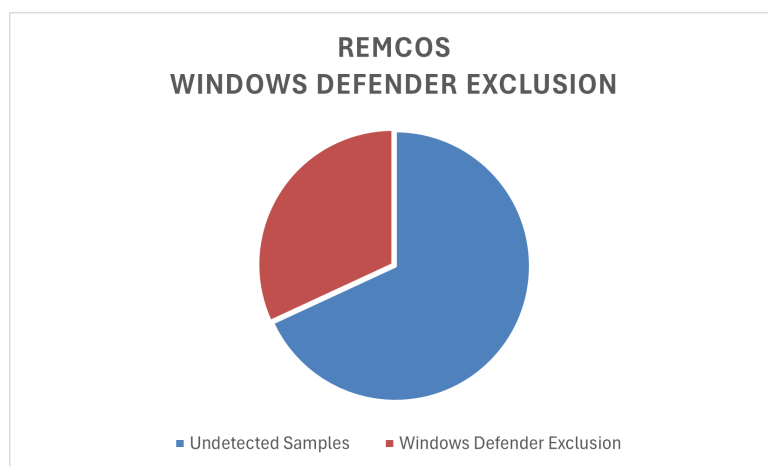
5.18: Redline Certificate

Due to the aforementioned fact, the JA4X could not be calculated, and the fingerprint value was NULL. The above configuration was considered an anomaly and can be used in conjunction with other information to created accurate detections for the malware.

5.2.3 Remcos Insights

Endpoint Analysis

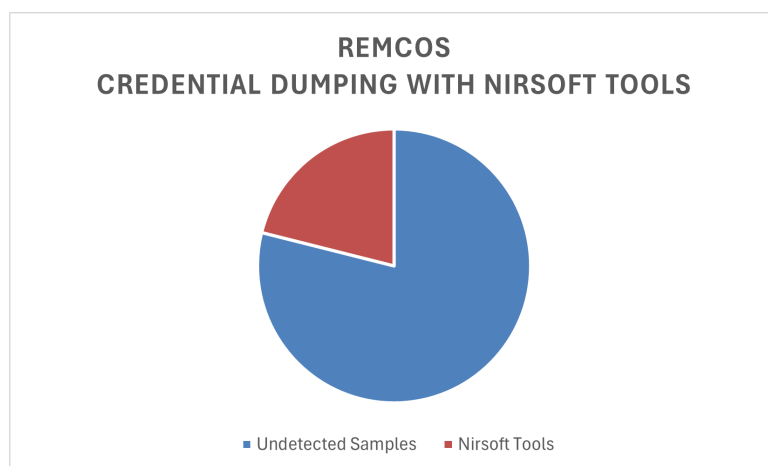
In the analysis of the TTPs that the Remcos malware performed on the endpoint, the technique Impair Defenses: Disable or Modify Tools was used by adding exclusions to the Defender. This technique had been highly utilized by all the malware families that were analyzed. In the case of the Remcos RAT family, it was utilized by 32% of the data, representing 59 of the functional samples. It is also worth mentioning that two Remcos samples were also observed using base64-encoded commands to add exclusions to Windows Defender.



5.19: Remcos - Windows Defender Exclusion

Another technique performed by all of the analyzed malware was spawning and injecting into legitimate files to avoid detections. Contrary to the other families that highly utilized this, only twelve Remcos samples (6%) were identified performing such activities.

An important functionality discovered during the malware analysis was the use of Nirsoft products to dump browser credentials. NirSoft is a legitimate software publisher that offers free Windows utility tools for purposes such as administration and forensic analysis. By using the command `DumpBrowserHistoryUsingNirsoft`, the malicious actor can download the Nirsoft tool from the C2 server, dump the browser's credentials, and store them in the temp directory.



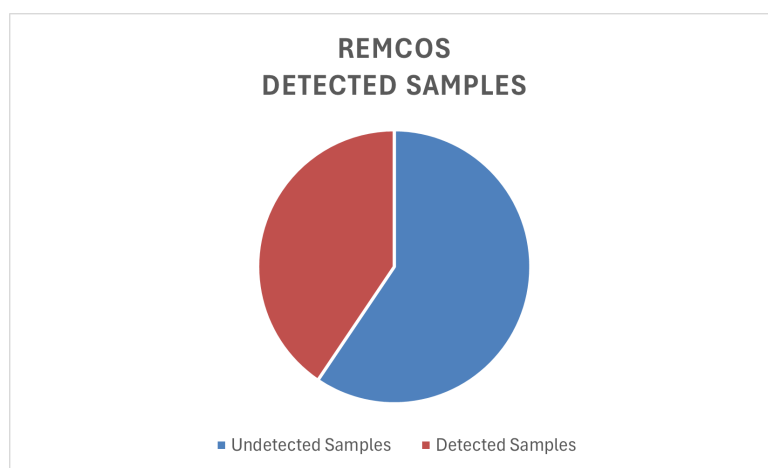
5.20: Remcos - Credential Dumping with Nirsoft Tools

Furthermore, a noteworthy observation was that some samples copied themselves to other directories, such as ProgramData or Appdata Roaming, using a file name containing the word Remcos. Although only five samples were observed, this finding could indicate with high confidence that a Remcos RAT is present on a host.

It is also important to note that the Remcos malware was identified performing persistence on the infected host by creating a scheduled task. The scheduled task was configured to execute a binary from the AppData directory with the flag onlogon, thus running the malware when a user logs in. It is worth mentioning that only one sample performed the aforementioned technique, thus indicating that it is not commonly used by the operators when the malware is executed.

Finally similarly, with the Redline family, Remcos samples were detected by renaming files with the move command and executing them using a cmd one-liner. It should be noted that only four samples performed the technique.

When summarizing the techniques performed by the Remcos family, each technique was used by a small number of samples, with the exception of Defense evasion via tampering the Windows Defender. By combining the aforementioned endpoint observations, 40% of the analyzed samples could have been identified.



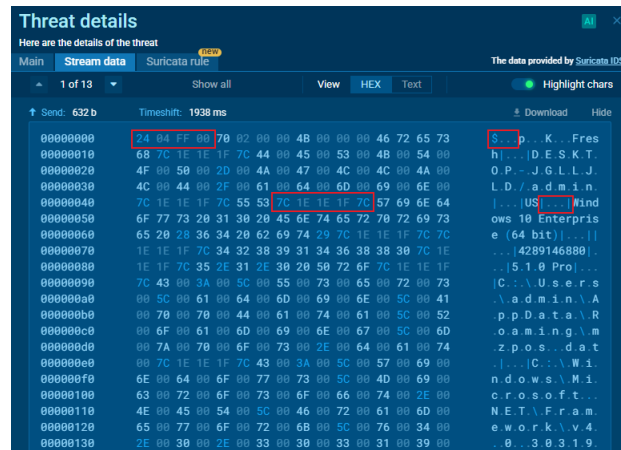
5.21: Remcos - Detected Samples

Network Analysis

Pivoting to the Network Telemetry, it was identified that Remcos utilized multiple different destination ports, mostly 80 and 443. Specifically, Remcos was detected using a custom protocol for C2 communication that is either unencrypted or secured with TLS 1.3, while attempting to blend in with the noise.

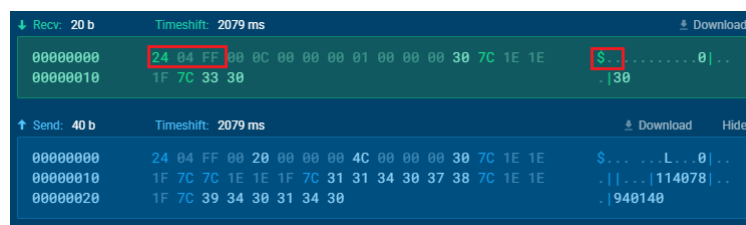
When querying the C2 servers on the Censys database, it was found that they were categorized as a product Remcos. This detection triggered when the certificate subject and issuer are empty and the JA4S has a unique value.

During the analysis of the unencrypted traffic, it was observed that inside each packet, the value 24 04 FF 00 was identified, which was the packet magic ID that Remcos is using. In a research performed by Fortinet, similar values were identified, with the most important being 7C 1E 1E 1F 7C, which corresponds to the string |. This character was used to split the information that Remcos has collected[20].



5.22: Remcos Client Unencrypted Traffic

The same packet magic ID, 24 04 FF 00, could be observed in the server's communication with the Remcos RAT.



5.23: Remcos Server Unencrypted Traffic

Similar to the Agent Tesla and Redline malware, Remcos was also identified communicating with the legitimate domain `geoplugin[.]net` to obtain information regarding the external IP of the infected host. This technique was identified in 39% of the analyzed functional malware.

JA4+ Fingerprints

Shifting the analysis to the encrypted communication and more specifically the JA4 fingerprint, three different values were observed.

JA4	Malware Count
t13i010400_0f2cb44170f4_5c4c70b73fa0	73
t12d190800_d83cc789557e_7af1ed941c26	8
t12d210700_76e208dd3e22_2dae41c691ec	2

Table 5.6: Remcos JA4

The JA4 fingerprint `t13i010400_0f2cb44170f4_5c4c70b73fa0`, which most of the samples used, corresponded to the encrypted Remcos traffic.

- Protocol: TCP
- TLS Version: 1.3
- SNI: Domain: IP
- Number of Ciphers: 01
- Number of Extensions: 04
- First ALPN: No ALPN

As can be observed, only 1 Cipher and four extensions are being used in the Client Hello packet, which was low compared to the other malware. The above observation makes the malware's custom protocol stand out, and the JA4 SSL fingerprints valuable artifacts to detect it.

After further investigation, it was concluded that the other two fingerprints belong to two different loader malware. This behavior was expected since the Remcos RAT is a heavily signed tool used for Command and Control and usually is not part of the initial access. In such cases, loader malware are commonly used to avoid detection and execute the Remcos malware in memory.

It is worth noting that all of the samples with the JA4 `t12d190800_d83cc789557e_7af1ed941c26` correspond to the DBatLoader, also known as ModiLoader. This malware is a Delphi compiled binary which is known to use multi-layer obfuscation and image steganography techniques to hide the malware. From the other hand, after further analysis there was no clear indication of which droppers correspond to the JA4 fingerprint `t12d210700_76e208dd3e22_2dae41c691ec`.

Pivoting to the JA4S, the TLS Server Fingerprint, Remcos custom protocol also created the unique JA4S `t130200_1301_234ea6891581`. This was expected since the client uses a low number of Ciphers and Extensions. The above observations made the JA4 and JA4S fingerprints of the Remcos RAT very unique.

Expanding the investigation to the Certificate that Remcos used, it was concluded that all the C2 domains did not use an Issuer or a Subject. This also implies that the JA4X calculated during the Command and Control was empty.

5.2.4 Similarity Analysis Across Malware Families

Although the three malware families were different software with different functionalities, it is important to understand their similarities and, thus, the standard techniques employed by malware today. This comparison will be performed on the collected artifacts, specifically the Endpoint Artifacts, the Network Artifacts, and the JA4 fingerprints.

Cross-Family Malware Similarities

The most valuable findings were the ones utilized by all of the malware families. Regarding endpoint behavior, it was identified that all of the families tampered the Windows Defender by adding exclusions and performing Process Hollowing. These were both Defense Evasion techniques commonly used by malicious software. It is also important to note that adding exclusions in Defender demonstrates that the malware was utilized by attackers who were opting for straightforward methods rather than sophisticated and stealthy tactics to achieve their goals.

From the network side, a heavy utilization of legitimate websites was identified by all of the malware families. More specifically all of them had a module to perform an external IP lookup on a legitimate site.

Malware Name	Domain
Agent Tesla	api[.]ipify[.]org
RedLine	api[.]ip[.]sb
Remcos	geoplugin[.]net

Table 5.7: Malware and Associated Domains

This functionality can provide helpful information to the attacker but also serve as an anti-analysis technique. More specifically, malware analysts and sandboxes may analyze malicious samples in offline environments, thus reducing the risk of infection. To combat such techniques, malware may communicate with legitimate sites, and if no results were returned, it may stop its functionalities or delete itself.

Furthermore, malware samples utilized legitimate sites to communicate with their handler. Specifically, to avoid detection, some samples utilized applications such as Telegram to upload information and domains such as OneDrive to download data. It is important to note that although this technique was identified in all of the datasets, in the case of the Remcos RAT, malware loaders utilized such domains to install the Remcos RAT and they were not used by Remcos directly.

Agent Tesla and Redline Similarities

When comparing the two infostealers, Agent Tesla and Redline, it was identified that both were written in C#. Threat actors often utilize this programming language since it is deeply integrated in Windows, provides access to the Windows API, and is also a high-level programming language,

thus making development easier. It is also worth mentioning that, due to the aforementioned facts, and since they use the same function to perform C2 communication, they had the same JA4 fingerprint, `t12d210700_76e208dd3e22_2dae41c691ec`.

Redline and Remcos Similarities

When comparing the Redline malware with the Remcos RAT, some interesting observations emerge. Firstly, regarding the endpoint artifacts, both families utilized Schedule Tasks as a persistence mechanism but very rarely. On top of that, they were both identified using an one-liner command during which the proper file extension was added to the file and then it was executed.

Another important similarity was that both malware families' SSL used in their C2 server has the Certificate Subject and Issuer values empty. This configuration could allow the tracking of malware infrastructure on the Internet and enabled the creation of detections utilizing the JA4X fingerprint.

5.2.5 Comparing Benign and Malicious JA4+ Fingerprints

5.3 Malware Detections

After analyzing the collected artifacts, it was important to transform them into intelligence, which could help respond to potential upcoming threats. These proactive measures can take multiple forms, but this research used detections. Detection in cyber security is a logic used to detect potential malicious or unauthorized activity.

The following detection focused on network telemetry using Suricata and endpoint telemetry using Sigma. This research tried to follow the principle of detection as code and not invest in a product's language but in generic logic.

5.3.1 Endpoint Detections

As previously mentioned, the Endpoint Detection was made using Sigma. Since multiple security vendors offer different detection options, Sigma was created to be a common language for writing detections. The most important part of a Sigma rule is the `"detection"`. Each Sigma rule is categorized and split into `"selections"`, which contain the definition for the detection itself. Inside the selections, there are `"fields"` that contain keywords that are being used to detect the activities according to the `"condition"`.

Defender Exclusion

One of the main techniques all malware families utilized was adding exclusions to Microsoft Defender. This is an easy technique used by malware to evade defense and bypass the Antivirus. Since this activity is not usually observed in an enterprise environment, a Sigma detection could provide high-value detections. The malware performed this by executing a Powershell and using the command-line arguments `Add-MpPreference` and `ExclusionPath` [1].

The Remcos RAT also executed base64 versions of the commands, which were added to the Sigma rule. It is important to note that due to the nature of the base64 encoding, variations

of the command could change the string used in this detection, thus bypassing it. Despite this, every Remcos sample that used the encoded version utilized the exact same string.

Argument	Description
Add-MpPreference	Modifies settings for Windows Defender.
ExclusionPath	Adds the selected folder to the exclusion list. The command disables Windows Defender scheduled and real-time scanning for files in this folder.

Table 5.8: PowerShell Defender Exclusion

```

title: Microsoft Defender File Path Exclusion
id: d2f5c9d8-3e7b-4a6c-92d1-1a9e8b5b76fc
status: Experimental
description: Detects the usage of the command Add-MpPreference to exclude
            file paths from Defender's real time scans.
author: Christos Fotopoulos
references:
  - https://learn.microsoft.com/.../add-mpreference
tags:
  - attack.defense-evasion
  - attack.T1562.001
logsource:
  category: process_creation
  product: windows
detection:
  selection_powershell:
    Image|endswith:
      - '\\powershell.exe'
      - '\\pwsh.exe'
  selection_cmdline:
    CommandLine|contains|all:
      - 'Add-MpPreference'
      - '-ExclusionPath'
  selection_cmdline_base64_AddMpPreference:
    CommandLine|contains:
      - QQBkAGQALQBNAHAAUABYAGUAZgB1AHIAZQBUBuAGMAZQ
  selection_cmdline_base64_ExclusionPath:
    CommandLine|contains:
      - COARQB4AGMABAB1AHMAaQBvAG4AUABhAHQAaa
  condition: selection_powershell and (selection_cmdline or all of
            selection_cmdline_base64_*)
falsepositives:
  - Administrator activities
level: High

```

Listing 5.1: Sigma rule for Defender Exclusions

Suspicious Filename

Another rule that can provide fruitful results was the execution of suspiciously named binaries. More specifically, Redline samples were observed executing one-character files named 1.exe, and Remcos samples ran files with "remcos" in their names. Although the percentage of such samples was low, the detections were noteworthy due to their low number of false positives. It is also worth mentioning that threat actors commonly utilize one-character files during later stages of their attack.

```
title: Executable Named Remcos
id: 1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d
status: Experimental
description: Execution of a exe named remcos.
references:
  - https://app.any.run/tasks/a4f3d879-4b75-44a0-9278-ed9451f3b750
author: Christos Fotopoulos
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    Image|endswith: "remcos.exe"
  condition: selection
falsepositives:
  - Unknown
level: High
```

Listing 5.2: Sigma rule for Executable Named Remcos

```
title: One Character Executable
id: 1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d
status: Experimental
description: Execution of a file with name lenght equal to one.
references:
  - https://app.any.run/tasks/957de006-78a4-4a64-8b99-0238983b5f4a
author: Christos Fotopoulos
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    Image|re: "\\b[A-Za-z0-9]\\b\\.exe\\b"
  condition: selection
falsepositives:
  - Unknown
level: High
```

Listing 5.3: Sigma rule for One Character Executable

Browser Credentials Dumping

One feature of the Remcos RAT was the ability to dump the browser's credentials using legitimate Nirsoft tools. While executing this plugin, the malware spawned a new instance of itself using

the argument `/text` and then specifies the directory where the information was stored.

Command Line
C:\Users\user\Desktop\Files\ ae284655948354c6ed48e95cf2aaa 058d376ed19d2aa69aa38eecea72ee2f576.exe /stext "C:\Users\user\AppData\Local\Temp\ cgrqchlwmnchrzjpnzqthowrvku"

Table 5.9: Dumping Browser Credentails with Nirsoft

```

title: Nirsoft Tool Injection
id: a1b2c3d4-e5f6-7890-g1h2-i3j4k5l6m7n
status: Experimental
description: RemcosRAT injects Nirsoft tools and uses the flag /sext in
            order to dump passwords.
author: Christos Fotopoulos
references:
  - https://www.elastic.co/security-labs/dissecting-remcos-rat-part-
    three
tags:
  - attack.attack.credential-dumping
  - attack.T1003
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    Image|endswith:
      - ".exe"
    CommandLine|contains|all:
      - '/stext'
  condition: selection
falsepositives:
  - Use of /sext flag from legitimate tools.
level: Medium

```

Listing 5.4: Sigma rule for Nirsoft Tool Injection

Persistence via Scheduled Task

Furthermore, the malware families Redline and Remcos utilized persistence via scheduled tasks. Adversaries may use task scheduling to execute programs at system startup or on a scheduled basis for persistence. Although this technique was not heavily used during the malware's execution, it can be valuable due to its heavy utilization during the later stages of the infection [13].

During the scheduled task creation, two anomalies could be observed. Firstly, the creation of a scheduled task that executes processes from the AppData Roaming, a common folder used by malware authors. Except for that, the creation of scheduled tasks with long duration was observed. It is important to note that a requirement for the flag `onlogon` was added to reduce the false positive detection. This option enabled the execution of the scheduled task when a user logs on to the workstation.

```
title: Schedule Tasks from Appdata/Roaming
id: g7h8i9j0-k1l2-3456-m7n8-o9p0q1r2s3
status: Experimental
description: Detects schedule tasks that execute files from the appdata
            directory.
author: Christos Fotopoulos
references:
  - https://attack.mitre.org/techniques/T1053/005/
tags:
  - attack.persistence
  - attack.T1053.005
logsource:
  category: process_creation
  product: windows
detection:
  selection_schtasks:
    Image|endswith:
      - '\schtasks.exe'
  selection_create_flag:
    CommandLine|contains|all:
      - '/create'
      - 'onlogon'
  selection_appdata_roaming:
    CommandLine|re: 'C:\\Users\\.*\\AppData\\Roaming'
  condition: all of selection_*
falsepositives:
  - Legitimate schedule tasks from the appdata directory.
level: Medium
```

Listing 5.5: Sigma rule for Schedule Tasks from Appdata/Roaming

```

title: Unusual Duration of Schedule Task
id: h8i9j0k1-l2m3-4567-n8o9-p0q1r2s3t4
status: Experimental
description: Detects the creation of a schedule task that will be run for
            a long duration.
author: Christos Fotopoulos
tags:
  - attack.persistence
  - attack.T1053.005
logsource:
  category: process_creation
  product: windows
detection:
  selection_schtasks:
    Image|endswith:
      - '\schtasks.exe'
  selection_create_flag:
    CommandLine|contains|all:
      - '/create'
      - 'onlogon'
  selection_duration:
    CommandLine|re: '/du\s+9999'
  condition: all of selection_*
falsepositives:
  - Legitimate schedule tasks.
level: Medium

```

Listing 5.6: Sigma rule for Unusual Duration of Schedule Task

Process Hollowing

The final detection created was Process Hollowing. Process hollowing was commonly performed by creating a process in a suspended state and then unmapping/hollowing its memory, which can then be replaced with malicious code. To detect this technique, a combination of two different Sigma rules was needed.

The first Sigma rule detected the execution of a process without arguments. During the process hollowing technique, it was common to execute the process without arguments since they were not needed in the attack. Consequently, this fact could be leveraged to create a rule that detected processes that should include command-line arguments but do not have them in the analyzed log. The process most commonly utilized by malware families during this research was `MsBuild.exe`, and thus, the sample detection was created using this file.

```

title: Process Hollowing
id: 1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d
status: Experimental
description: Execution of legitimate process without parameters.
author: Christos Fotopoulos
logsource:
  product: windows
  service: sysmon
detection:
  selection_msbuild:
    EventID: 1
    Image|endswith: "\\MsBuild.exe"
  selection_cmdline:
    EventID: 1
    CommandLine|endswith: #no flags are being used
      - '\\MsBuild.exe"' #sysmon wrapped the cmdline with "
    condition: all of selection_*
level: High

```

Listing 5.7: Sigma rule for Process Hollowing

The second rule created was the detection of this process in an Event ID three sysmon log, which indicates that it performed a network connection. One of the most common goals of a malicious actor is to execute their agents to achieve command and control, thus, they tend to perform process hollowing, inject a malware agent, and then the process performs network connections. This Sigma rule is optional and can be helpful in environments where the first detection generates many false positives.

```

title: Process Hollowing
id: 1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d
status: Experimental
description: Network connection of legitimate process without parameters.
references:
  - https://app.any.run/tasks/957de006-78a4-4a64-8b99-0238983b5f4a
author: Christos Fotopoulos
logsource:
  product: windows
  service: sysmon
detection:
  selection_network_connection:
    EventID: 3
    Image|endswith: "\\MsBuild.exe"
    condition: all of selection_*
falsepositives:
  - Unknown
level: High

```

Listing 5.8: Sigma rule for Process Hollowing

Although Sigma rules do not support the combination of the above detections, it is recommended that they be combined during their conversion to the desired querying language.

5.3.2 Network Detections

Even though network artifacts provide less visibility than endpoint artifacts, they were still an important angle that should be explored. Network detections could complement the endpoint rules to fill gaps or provide additional information. The importance of network detections was also highly showcased in environments that could achieve network visibility to all workstations or during attacks on edge devices.

Edge devices are hardware such as Firewalls, VPNs, and Load balancers. They are typically placed at the boundary of a network. Due to their nature and typically custom OS, monitoring their endpoint activities can be impossible. Mandiant, in M-Trends 2025 report, said that according to their investigations conducted in 2024, the most frequently exploited vulnerabilities typically target edge devices.[34]

In this research, Suricata rules were used to create the network detections. Suricata is the most common language used by the research community for such purposes and is also supported by multiple tools, such as the Suricata IPS and the Corelight NDR.

Agent Tesla Detections

As mentioned in the section related to Agent Tesla's Network traffic analysis, during data exfiltration, specific arguments were used. More specifically, during the use of unencrypted SMTP and FTP protocols, the exfiltrated data was structured using the following pattern:

- Time:
User Name:
Computer Name:

Based on the above, in order to detect the unencrypted exfiltration using FTP and SMTP, the aforementioned pattern were used. On top of that, to improve the rule's performance, the protocol TCP and the traffic's direction from internal to external were specified.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any
(msg:"AgentTesla Unencrypted FTP/SMTP Exfiltration";
flow:established,to_server;
content:"Password";
pcree:"/Time:.*?<br>User Name:.*?<br>Computer Name:/" ;
offset:370; depth:110;
sid:100001; classtype:command-and-control; priority:1;
reference:url,https://corelight.com/blog/detecting-agent-tesla-malware;)
```

Listing 5.9: Suricata rule for AgentTesla Unencrypted FTP/SMTP Exfiltration

Redline Detections

During the analysis of the Redline malware, the usage of the SOAPAction containing the value tempuri.org/Endpoint/ <Option > was observed. In conjunction with the condition of the method POST and the protocol HTTP, this observation was combined to create the following detection.

Option	Description
tempuri.org/Endpoint/CheckConnect	Verify if the connection is functioning properly
tempuri.org/Endpoint/EnvironmentSettings	Requests file to collect
tempuri.org/Endpoint/SetEnvironment	Uploads stolen data
tempuri.org/Endpoint/GetUpdates	Uploads stolen data and requests for additional tasks

Table 5.10: Detection of Redline C2 Options

```

alert http $HOME_NET any -> $EXTERNAL_NET any
(msg:"RedLine C2 Unencrypted SOAP";
 flow: established,to_server;
 http.header;
 pcre:"/SOAPAction:\s\"http://tempuri\.org/Endpoint/
(GetUpdates|CheckConnect|EnvironmentSettings|SetEnvironment)\"/";
 http.method;
 content:"POST";
 sid:100002;
 classtype:command-and-control;
 priority:1;
 reference:url,https://www.fortinet.com/blog/threat-research/excel
-document-delivers-multiple-malware-exploiting-cve-2017-11882-part-
two;)

```

Listing 5.10: Suricata rule for RedLine C2 Unencrypted SOAP

Similarly, with the above detection, Redline was identified containing the string `tempuri.org/Entity/Id<0-9>` inside its Command and Control traffic packets. Based on that the following Suricata rule was created:

```

alert tcp any any -> any any (
 msg:"RedLine C2 Unencrypted";
 content:"http://tempuri.org/Entity/Id";
 pcre:"/http://tempuri\.org/Entity/Id[0-9]+/";
 sid:100001;
 classtype:command-and-control;
 priority:1;
 reference:url,https://www.cloudsek.com/blog/https://www.cloudsek.com/
blog/technical-analysis-of-the-redline-stealer;
)

```

Listing 5.11: Suricata rule for RedLine C2 Unencrypted

The final two Suricata detections were created to identify anomalies in Redline's encrypted communication. More specifically, the suspicious SNI with the value `?124` and the certificate with the subject and issuer value equal to `"CN=localhost"` was utilized to detect the encrypted C2 communication.

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (
  msg:"RedLine C2 Suspicious SNI";
  flow:established,to_server, only_stream;
  content:"|00 07 00 00 04 3f|124|00 0a|";
  sid:100004;
  classtype:command-and-control;
  priority:1;
  reference:url,https://community.emergingthreats.net/t/
  metastealer-v-5-tls/1800;
)

```

Listing 5.12: Suricata rule for RedLine C2 Suspicious SNI

```

alert tcp $EXTERNAL_NET any -> $HOME_NET any (
  msg:"RedLine C2 Localhost Certificate";
  flow:established,to_client;
  content:"|30 14 31 12 30 10 06 03 55 04 03 13 09|localhost";
  offset:145; depth:22;
  content:"|30 14 31 12 30 10 06 03 55 04 03 13 09|localhost";
  distance:30; within:60;
  sid:100005;
  classtype:command-and-control;
  priority:1;
  reference:url,https://www.fortinet.com/blog/threat-research/
  excel-document-delivers-multiple-malware-exploiting-cve-2017-11882-
  part-two;
)

```

Listing 5.13: Suricata rule for RedLine C2 Localhost Certificate

It is also worth mentioning that detections created by emerging threats regarding the aforementioned activity were identified. These rules have additional conditions to decrease false positive detection and, therefore, are more suitable for a production environment.

```

alert tcp $EXTERNAL_NET any -> any any (
  msg: "ET MALWARE [ANY.RUN] MetaStealer v.5 (MC-NMF TLS Server
    Certificate)";
  flow: established, to_client;
  content: "|0a160303|"; depth:4;
  content: "|30 82 01 25 30 81 d0 a0 03 02 01 02 02 10|"; distance:0;
  content: "|30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00 30 14 31 12
    30 10 06 03 55 04 03 13 09 6c 6f 63 61 6c 68 6f 73 74 30 1e 17 0d|
    ";
  distance:16; within:45;
  threshold: type both, track by_dst, seconds 360, count 1;
  classtype: command-and-control;
  reference:url,https://community.emergingthreats.net/t/metastealer-v
    -5-tls;
  metadata: created_at 2024_07_08, tag stealer, malware_family
    MetaStealer;
  sid:1; rev:1;
)

```

Listing 5.14: Emerging Threats Suricata rule for MetaStealer Server Certificate

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (
  msg:"ET MALWARE [ANY.RUN] MetaStealer v.5 CnC Activity (MC-NMF TLS
    SNI)";
  flow:established,to_server, only_stream;
  content: "|00 01 00 01 02 02 1f|net.tcp://"; startswith;
  content: "|03 08 09 13|application/ssl-tls"; distance: 0;
  content: "|00 07 00 00 04 3f|124|00 0a|"; distance: 0; within: 200;
  fast_pattern;
  reference:url,https://community.emergingthreats.net/t/metastealer-v
    -5-tls/
  1800;
  classtype:trojan-activity;
  sid:2054404; rev:2;
  metadata: affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit,
    attack_target Client_Endpoint,
    tls_state plaintext,
    created_at 2024_07_08,
    deployment Perimeter,
    malware_family MetaStealer,
    confidence High,
    signature_severity Major,
    updated_at 2024_07_08;
)

```

Listing 5.15: Emerging Threats Suricata rule for MetaStealer CnC Activity

Remcos Detections

The final malware analyzed in this research was the Remcos RAT. When investigating the unencrypted traffic of this malware, two Suricata detections by Emerging Threat were discovered. As discussed in the chapter on Remco's network analysis, the malware utilizes a custom Command and Control protocol that used specific bytes. Based on this finding, the following detections were created for the unencrypted connection from the client and the unencrypted response of the server.

```
alert tcp-pkt $HOME_NET any -> $EXTERNAL_NET any (
  msg:"ET MALWARE Remcos 3.x Unencrypted Checkin";
  flow:established,to_server;
  content:"|24 04 ff 00|"; startswith;
  content:"|4b 00 00 00|"; distance:4; within:4;
  content:"|7c 1e 1e 1f 7c|"; distance:0;
  fast_pattern;
  reference:md5,d27f70216d11b769c937a961fc1b1c81;
  classtype:command-and-control;
  sid:2032776; rev:2;
  metadata: attack_target Client_Endpoint,
            created_at 2021_04_16,
            deployment Perimeter,
            malware_family Remcos,
            performance_impact Low,
            confidence High,
            signature_severity Major,
            updated_at 2021_04_16;)
```

Listing 5.16: Emerging Threats Suricata rule for Remcos Unencrypted Client Checkin

```
alert tcp-pkt $EXTERNAL_NET any -> $HOME_NET any (
  msg:"ET MALWARE Remcos 3.x Unencrypted Server Response";
  flow:established,to_client;
  content:"|24 04 ff 00|"; startswith;
  content:"|01 00 00 00 30 7c 1e 1e 1f 7c|"; distance:4; within:10;
  fast_pattern;
  threshold: type limit, track by_src, count 1, seconds 120;
  reference:md5,d27f70216d11b769c937a961fc1b1c81;
  classtype:command-and-control;
  sid:2032777; rev:2;
  metadata: affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit,
            attack_target Client_Endpoint,
            created_at 2021_04_16,
            deployment Perimeter,
            malware_family Remcos,
            confidence Medium,
            signature_severity Major,
            updated_at 2021_04_16;)
```

Listing 5.17: Emerging Threats Suricata rule for Remcos Unencrypted Server Response

5.3.3 JA4+ Detections

The final category of detections used JA4+ fingerprints. Currently, there was no detection language or technology for the fingerprints, so the logic was written using SQL. It is important to note that a malicious actor could theoretically spoof the JA4+ fingerprints, but there have not been any reports of this happening yet therefore the detections could create important results.

Agent Tesla Detections

After analyzing Agent Tesla's JA4+ fingerprints, it was concluded that no anomalies could be identified. However, a detection was made for the encrypted exfiltration with SMTP protocol.

As previously discussed, Agent Tesla only had one JA4 fingerprint with the value `t12d210700_76e208dd3e22_2dae41c691ec`. On the other hand, multiple JA4S fingerprints were used. The difference was the number of extensions, which varies from five to two. Furthermore, two variations of ciphers were used. The first variation had the value `c030` and was identified in the traffic of 87.5% of the malware. The second had the value `c02f` and was detected at 12.5%.

Also, after examining the JA4X fingerprints, it was identified that the `a373a9f83c6b_7022c563de38_821a8ec155c6` and `2bab15409345_7022c563de38_0321e8b32200` correspond to 84% of the Agent Tesla fingerprints.

Depending on an organization's traffic, the values of JA4S_b and JA4X can be specified if there is a need to reduce false positives, but this may lead to missing activities from Agent Tesla samples. More specifically, when adding all the filters, only 38 samples were detected, corresponding to 31% of the dataset. In conclusion, since there were no anomalies in the SMTP communication of the Agent Tesla malware, there can not be a detection that works in all environments. Despite that, it is important to note that this detection should not be ignored. For example, an organization that uses SMTP but the applications do not have the aforementioned JA4 can deploy this detection with a low false positive rate.

```
SELECT MalwareHashID
FROM "JA4 Artifacts"
WHERE dstport = '587'
  AND JA4 = 't12d210700_76e208dd3e22_2dae41c691ec'
  AND JA4S LIKE 't120%00_c030_%'
  AND (
    JA4X LIKE 'a373a9f83c6b_7022c563de38_%' OR
    JA4X LIKE '2bab15409345_7022c563de38_%')
GROUP BY MalwareHashID;
```

Listing 5.18: JA4+ Detection for Agent Tesla

Redline Detections

In contrast with Agent Tesla, important anomalies were identified during the Redline's TLS handshake. As previously discussed, although they had the same JA4 fingerprint, Redline had only one JA4S for its C2 communication. On top of that, the Server Name Indication (SNI) had the unique value ?124. Finally, another important observation is that the JA4X has the value NULL. Based on the above, the following detection could be created, which has many singularities and thus should have fewer false positive detections.

```
SELECT MalwareHashID
FROM "JA4 Artifacts"
WHERE domain = '?124'
  AND JA4 = 't12d210700_76e208dd3e22_2dae41c691ec '
  AND JA4S = 't120200_c030_5333cdffa7d9 '
  AND "JA4X" IS NULL
GROUP BY MalwareHashID;
```

Listing 5.19: JA4+ Detection for Redline

Remcos Detections

The final malware analyzed is the Remcos RAT. This malware utilized a custom protocol, and during its analysis, it was observed that it utilizes only one cipher and four extensions. On top of that, only two extensions were selected by the server, and since the client offered only one cipher, that cipher was the one ultimately selected. Finally, it is important to note that as, was previously discussed, the JAX fingerprint was once again empty due to the uniqueness of the certificates. The above facts created unique JA4+ fingerprints and thus a rule could be created with a high detection rate and low amount of false positives.

```
SELECT MalwareHashID
FROM "JA4 Artifacts"
WHERE JA4 = 't13i010400_0f2cb44170f4_5c4c70b73fa0 '
  AND JA4S = 't130200_1301_234ea6891581 '
  AND "JA4X" IS NULL
GROUP BY MalwareHashID;
```

Listing 5.20: JA4+ Detection for Remcos

Chapter 6

Discussion

6.1 Limitations

This research utilized Terraform to deploy virtual machines and create the sandboxes automatically. Then, it used Ansible playbooks to interact with them, execute malware, and collect the generated artifacts. Although the research utilized custom-made or open-source tools to achieve the above, there were still limitations that could have further improved its results.

One of this researcher's limitations was the hardware resources, which had 10 RAM and 4 CPU cores. These specifications allowed only three virtual machines to be created at a time, each with 2 GB of RAM and 1 CPU core. Although the resources were sufficient to perform the malware analysis, having access to high-performance computing infrastructure would offer additional advantages. More specifically, a malware sample could enumerate the resources to understand that they are too limited, which suggests that it is inside a sandbox, thus stopping its execution. The increase in resources could also allow the creation of more virtual machines simultaneously and thus enable the analysis of even more malware families.

In addition to that, the analyzed malware samples were a constraint during this research. Malicious samples of Agent Tesla, Redline, and Remcos families were executed each day. A daily selection of samples uploaded to Malware Bazaar the previous day was used to ensure that the samples had active and operational infrastructure that had not yet been taken down. More malware datasets, preferably with non-publicly available samples, would have provided even better results because the IOCs and the TTPs would not have been publicly available, and the infrastructure would also be functional.

6.2 Future Work

Since the created workflow aims to analyze malware samples, collect artifacts, and create intelligence, an objective for the future is ingesting more logs and creating more plugins. More specifically, there are many more logs, such as Registry modifications, File creation and deletion, which can increase the visibility and provide even better results in the research. On top of that, creating new intelligence collection plugins, such as memory dumping, which would allow scanning it with Yara, would provide even more fruitful results.

Another development aim is to improve the current workflow, making it more efficient and user-friendly. More precisely, this tool's generated artifacts are excessive in size and, depending on the hardware resources, more samples could be analyzed making the intelligence creation even harder. To tackle that problem, automation playbooks and large language models can help the analyst investigate the data more easily and faster. Except for that, there is a need for improvement in the tool's UI and UX. Although it has been built to allow easy customization, there is a need for a UI interface to allow users to make changes to the workflow without needing to interact with the source code.

Finally, an important goal for the future is testing the created detections on larger datasets. In particular, commercial platforms such as Anyrun and VirusTotal allow the deployment of Sigma, Suricata, and JA4 detections on their large datasets. By doing so, testing could be performed regarding their detection rate and false positives. On top of that, some platforms provide malware configuration extraction. The configuration of a malware sample defines how the malware behaves and can provide C2 domains, campaign IDs, and further information to track the campaigns and the malware operators.

Chapter 7

Conclusion

This research focused on a nontraditional approach to consuming threat intelligence. Instead of parsing intelligence feeds, it utilized custom Sandboxes created to allow scalability and customization. This approach allowed the simultaneous automatic dynamic analysis of multiple malware samples, thus producing many artifacts.

In the aforementioned automated workflow, malware samples from the families Agent Tesla, Redline, and Remcos were automatically executed. During this process, endpoint and network artifacts were collected in the form of Sysmon logs and packet captures. After the artifacts were collected to a centralized host, they were enriched with information from the platform Censys as well as using both open-source tools such as JA4 and custom tools.

Once the automated analysis was concluded, manual analysis was performed to create intelligence. Five different MITRE techniques and seven C2 communication methods were identified from the analysis of the malware families. It is also important to note that twenty different detections were made from the malware behavior: eight Sigma rules, nine Suricata including four signatures from the Emerging Threats list, and three using the JA4+ fingerprints.

Although parsing and ingesting intelligence is critical, it can be difficult and expensive. The research proposes a different approach that utilized sandboxes to analyze malware and thus automatically create intelligence.

Bibliography

- [1] “Add-MpPreference,” <https://learn.microsoft.com/en-us/powershell/module/defender/add-mppreference?view=windowsserver2025-ps>, [Online; accessed 15-May-2025].
- [2] “Ansible,” <https://www.redhat.com/en/ansible-collaborativeansible>, [Online; accessed 15-May-2025].
- [3] “Censys,” <https://about.censys.io/>, [Online; accessed 15-May-2025].
- [4] “HashiCorp Terraform,” <https://www.terraform.io/>.
- [5] “MalwareBazaar,” <https://bazaar.abuse.ch/>, [Online; accessed 15-May-2025].
- [6] “MITRE ATT&CK Framework,” <https://attack.mitre.org/>, [Online; accessed 15-May-2025].
- [7] “Remcos Sale Page,” <https://breakingsecurity.net/remcos/>, [Online; accessed 15-May-2025].
- [8] “TLS Handshake,” <https://www.cloudflare.com/en-gb/learning/ssl/what-happens-in-a-tls-handshake/>, [Online; accessed 15-May-2025].
- [9] “TLS Server Hello,” <https://www.ibm.com/docs/en/ibm-mq/9.3.x?topic=tls-overview-ssl-tls-handshake>, [Online; accessed 15-May-2025].
- [10] “TLS,” https://en.wikipedia.org/wiki/Transport_Layer_Security, 2003, [Online; accessed 15-May-2025].
- [11] “Proxmox Hypervisor,” <https://www.proxmox.com/en/>, 2004, [Online; accessed 15-May-2025].
- [12] “HTTP Protocol,” <https://en.wikipedia.org/wiki/HTTP>, 2010, [Online; accessed 15-May-2025].
- [13] “MITRE Scheduled Task/Job,” <https://attack.mitre.org/techniques/T1053/>, 2017, [Online; accessed 15-May-2025].
- [14] “MITRE Command and Control,” <https://attack.mitre.org/tactics/TA0011/>, 2018, [Online; accessed 15-May-2025].
- [15] “Impair Defenses: Disable or Modify Tools,” <https://attack.mitre.org/techniques/T1562/001/>, 2020, [Online; accessed 15-May-2025].
- [16] “Process hollowing being used by agent tesla,” <https://www.sentinelone.com/labs/agent-tesla-old-rat-uses-new-tricks-to-stay-on-top/>, 2020, [Online; accessed 15-May-2025].

- [17] “Process Injection: Process Hollowing,” <https://attack.mitre.org/techniques/T1055/012/>, 2020, [Online; accessed 15-May-2025].
- [18] “Redline SOAP Cyberint,” <https://cyberint.com/blog/research/redline-stealer/>, 2021, [Online; accessed 15-May-2025].
- [19] “Redline SOAP Fortinet,” <https://www.fortinet.com/blog/threat-research/excel-document-delivers-multiple-malware-exploiting-cve-2017-11882-part-two>, 2022, [Online; accessed 15-May-2025].
- [20] “Remcos RAT Analysis,” <https://www.fortinet.com/blog/threat-research/latest-remcos-rat-phishing>, 2022, [Online; accessed 15-May-2025].
- [21] “JA4 Blog,” <https://blog.foxio.io/ja4+-network-fingerprinting>, 2023, [Online; accessed 15-May-2025].
- [22] “JA4 Github,” <https://github.com/FoxIO-LLC/ja4>, 2023, [Online; accessed 15-May-2025].
- [23] “Redline and Metastealer,” <https://russianpanda.com/MetaStealer-Redline-s-Doppelganger>, 2023, [Online; accessed 15-May-2025].
- [24] “Redline HTTP Medium,” https://medium.com/@the_abjuri5t/advice-for-catching-a-redline-stealer-dca126867193, 2023, [Online; accessed 15-May-2025].
- [25] “Agent Tesla Detections,” <https://corelight.com/blog/detecting-agent-tesla-malware>, 2024, [Online; accessed 15-May-2025].
- [26] “Agent Tesla Detections,” <https://app.any.run/tasks/f9421792-7d2c-47d3-90e0-07eb54ae12fa/>, 2024, [Online; accessed 15-May-2025].
- [27] “Agent Tesla Detections,” <https://app.any.run/tasks/0e328ab7-12b2-4843-8717-a5b3ebef33a8>, 2024, [Online; accessed 15-May-2025].
- [28] “Agent Tesla Malware,” <https://www.malwation.com/blog/origins-of-a-logger-agent-tesla>, 2024, [Online; accessed 15-May-2025].
- [29] “Evolution of Cyber Attacks IBM,” <https://www.ibm.com/think/insights/decade-global-cyberattacks-where-they-left-us>, 2024, [Online; accessed 15-May-2025].
- [30] “Sysmon - SysInternals,” <https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>, 2024, [Online; accessed 15-May-2025].
- [31] “Werfault,” <https://www.python.org/>, 2024, [Online; accessed 15-May-2025].
- [32] “Evolution of Cyber Attacks SentinelOne,” <https://www.sentinelone.com/cybersecurity-101/cybersecurity/cyber-security-statistics>, 2025, [Online; accessed 15-May-2025].
- [33] “Malware Analysis Definition,” <https://www.crowdstrike.com/en-us/cybersecurity-101/malware/malware-analysis/>, 2025, [Online; accessed 15-May-2025].
- [34] “Mandiant M-Trends 2025,” <https://services.google.com/fh/files/misc/m-trends-2025-en.pdf>, 2025, [Online; accessed 15-May-2025].
- [35] “MITRE Masquerading,” <https://attack.mitre.org/techniques/T1036/>, 2025, [Online; accessed 15-May-2025].

- [36] M. T. Alam, D. Bhusal, Y. Park, and N. Rastogi, “Looking beyond iocs: Automatically extracting attack patterns from external cti,” in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 92–108. [Online]. Available: <https://doi.org/10.1145/3607199.3607208>
- [37] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, and M. van Eeten, “A different cup of TI? the added value of commercial threat intelligence,” in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 433–450. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity20/presentation/bouwman>
- [38] A. Djenna, A. Bouridane, S. Rubab, and I. M. Marou, “Artificial Intelligence-Based Malware Detection, Analysis, and Mitigation,” *Symmetry*, vol. 15, no. 3, p. 677, Mar. 2023.
- [39] S. Jamalpur, Y. S. Navya, P. Raja, G. Tagore, and G. R. K. Rao, “Dynamic malware analysis using cuckoo sandbox,” in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 1056–1060.
- [40] R.-V. Mahmoud, M. Anagnostopoulos, S. Pastrana, and J. M. Pedersen, “Redefining malware sandboxing: Enhancing analysis through sysmon and elk integration,” *IEEE Access*, vol. 12, pp. 68 624–68 636, 2024.