INTERNATIONAL HELLENIC UNIVERSITY

DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

THESIS

«Analysis of Malware Persistence Techniques»

Created with Bing AI Image Creator

Of Georgios Spyropoulos
Registration number: 144354

Supervisor
Full name: Christos Ilioudis
Rank: Professor

Date September 2024

Title: Analysis of Malware Persistence Techniques
Thesis Code: 23159
Student Name: Georgios Spyropoulos
Professor Name: Christos Ilioudis
Assignment Date: March 2023
End Date: September 2024

I hereby declare that I am the author of this work and that any assistance I have received in preparing it is fully acknowledged and referred to in the work. I have also recorded any sources from the reports I have used like data, ideas, images and text, whether they are accurate or paraphrased. In addition, I hereby declare that this thesis was prepared by me personally in the Department of Information and Electronic Engineering of I.H.U.

Prologue

The choice to focus my dissertation on the "Analysis of Malware Persistence Techniques" stems from my current role as a cybersecurity professional, specifically as a SOC Analyst. This subject aligns perfectly with my career aspirations to advance into the Incident Response team, where a deep understanding of malware behavior is crucial. My keen interest in how malware persists in systems motivated me to explore this topic, as mastering these techniques is essential for detecting potentially undetectable malware during my daily analyses at work.

Through this research, I have gained a comprehensive understanding of the various persistence techniques employed by different malware families. This knowledge has not only expanded my expertise in cybersecurity but also equipped me with the skills necessary to recognize and mitigate sophisticated threats in real-world scenarios. The insights gained have significantly contributed to my professional development, enhancing my ability to protect systems more effectively against persistent threats.

Abstract

This dissertation outlines various techniques that can be used by malware and how they can be used to establish a good foothold in the affected computers. The paper analyses the different persistence techniques that have been employed in malware over the years, the early and basic methods like registry hack and scheduled task, DLL side loading, and the more contemporary fileless malware. Using the samples downloaded from the VirusShare, the Zoo, and MalwareBazaar sources and analyzed with VirusTotal, the study adds the classification of a set of malware types and demonstrates the frequently used persistence techniques present in modern threats.

The research thus demonstrates that some of these strategies work; for instance, hijacking of execution flows and manipulation of startup processes for malware presence. Furthermore, the study also looks at the relationship between persistence strategies and other types of behaviors like defense evasion and command and control (C2) to achieve a deeper time understanding of how persistence is used in the real attacks.

The dissertation also reveals that there are major gaps in knowledge of existing detection techniques and how they manage to fail in the detection of fileless malware, APTs, and polymorphic malware. The study is intended both to expand the theoretical and practical understanding of cybersecurity threats and to introduce prospective research perspectives that will help to construct stronger defenses.

This research is of most interest to cybersecurity specialists who would like to get deeper insights into the work of malware and refine their approach to counteracting threats that use persistence mechanisms. The integration of the findings with the MITRE ATT&CK framework provides practical implementation of insights learned in the research study, with the bonus of being a useful reference study for both academics and practitioners in the field.

“Ανάλυση τεχνικών επιμονής κακόβουλου λογισμικού”

Σπυρόπουλος Γεώργιος

Περίληψη στα Ελληνικά

Αυτή η διατριβή παρέχει μια ολοκληρωμένη ανάλυση των τεχνικών επιμονής κακόβουλου λογισμικού, εστιάζοντας στον τρόπο με τον οποίο αυτές οι τεχνικές επιτρέπουν στο κακόβουλο λογισμικό να αποφύγει τον εντοπισμό και να διατηρήσει μια θέση σε παραβιασμένα συστήματα. Η έρευνα εξετάζει την εξέλιξη των μεθόδων εμμονής, από παραδοσιακές προσεγγίσεις όπως τροποποιήσεις μητρώου και προγραμματισμένες εργασίες έως πιο εξελιγμένες τεχνικές όπως η πλευρική φόρτωση DLL και το κακόβουλο λογισμικό χωρίς αρχεία. Αξιοποιώντας δεδομένα από πηγές όπως το VirusShare, το Zoo και το MalwareBazaar και αναλύοντας τα με το VirusTotal. Η μελέτη κατηγοριοποιεί και αναλύει μια τεράστια γκάμα δειγμάτων κακόβουλου λογισμικού, επισημαίνοντας τις πιο διαδεδομένες στρατηγικές εμμονής που χρησιμοποιούνται από τις σύγχρονες απειλές.

Τα ευρήματα υπογραμμίζουν την αποτελεσματικότητα ορισμένων τεχνικών, όπως η παραβίαση ροών εκτέλεσης και ο χειρισμός των διαδικασιών εκκίνησης, για τη διασφάλιση της επιμονής του κακόβουλου λογισμικού. Επιπλέον, η έρευνα διερευνά τη συσχέτιση μεταξύ τεχνικών επιμονής και άλλων τακτικών, όπως η αποφυγή άμυνας και οι επικοινωνίες εντολών και ελέγχου (C2), παρέχοντας μια λεπτή κατανόηση του τρόπου με τον οποίο αυτές οι μέθοδοι χρησιμοποιούνται σε επιθέσεις πραγματικού κόσμου.

Η διατριβή εντοπίζει επίσης σημαντικά κενά γνώσης στις τρέχουσες μεθοδολογίες ανίχνευσης, ιδιαίτερα στην αντιμετώπιση των προκλήσεων που τίθενται από κακόβουλο λογισμικό χωρίς αρχεία, προηγμένες επίμονες απειλές (APT) και πολυμορφικό κακόβουλο λογισμικό. Προτείνοντας μελλοντικές κατευθύνσεις έρευνας, η μελέτη στοχεύει να συμβάλει στη διαρκή ανάπτυξη πιο ισχυρών αμυντικών συστημάτων κυβερνοασφάλειας.

Αυτή η έρευνα είναι ιδιαίτερα σημαντική για τους επαγγελματίες της κυβερνοασφάλειας που επιδιώκουν να βελτιώσουν την κατανόησή τους για τη συμπεριφορά κακόβουλου λογισμικού και να βελτιώσουν τις στρατηγικές τους για τον εντοπισμό και τον μετριασμό των επίμονων απειλών. Η ενοποίηση των ευρημάτων με το πλαίσιο MITRE ATT&CK παρέχει περαιτέρω μια πρακτική εφαρμογή για τις γνώσεις που αποκτήθηκαν, προσφέροντας μια πολύτιμη πηγή τόσο για τους ακαδημαϊκούς όσο και για τους επαγγελματίες του κλάδου.

Acknowledgments

I would like to thank my professor for the guidance on my thesis and all those close to me who helped me in every way they could.

Table of Contents

Table of Figures

List of Abbreviations

Term	Abbreviation
Information Security	InfoSec
Command and Control	C&C
Distributed Denial-of-Service	DDoS
Intrusion Detection System	IDS
Intrusion Prevention System	IPS
Behavioral Analysis	BA
Static Analysis	SA
Dynamic Analysis	DA
Endpoint Detection and Response	EDR
Advanced Persistent Threat	APT
Antivirus	AV
Master Boot Record	MBR
Volume Boot Record	VBR

Unified Extensible Firmware Interface	UEFI
Internet of Things	IoT
Structured Query Language	SQL
Cyber Threat Intelligence	CTI
Virtual Machine	VM
Research Design	RD
Data Collection	DC
Secure Sockets Layer	SSL
Secure Hash Algorithm	SHA
Dynamic Link Library	DLL
Graphical User Interface	GUI
Credential Access Techniques	CAT
Cybersecurity	CS
Input Capture	IC

Operating System Credential Dumping	OSCD
Windows Management Instrumentation	WMI
Rescan Hashes	RH
Antivirus Vendors	AVV
Command & Control	C2
Hijack Execution Flow	HEF

1. Introduction

1.1. Background

The digital era has brought numerous advancements and improvements to our daily lives, but it has also introduced new challenges in information security. One of the most significant risks is malware, which cybercriminals use for various purposes, from data theft to system monitoring.

Malware encompasses all types of software designed to infiltrate or damage computer systems without the consent of the owner. It includes viruses, worms, Trojans, ransomware, and spyware among others. Understanding malware and its persistence techniques is crucial for modern cybersecurity.

Viruses are malicious programs that attach themselves to legitimate files and spread to other files. They often corrupt data or cause significant operational disruptions. Worms, on the other hand, are standalone malicious programs that replicate themselves to spread to other computers, often exploiting network vulnerabilities without needing to attach to other programs. Trojans disguise themselves as legitimate software but, once activated, execute malicious activities such as data theft or enabling unauthorized access to the user's system.

Ransomware has gained notoriety for its ability to encrypt a victim's files, demanding a ransom for the decryption key. This type of malware can cripple organizations by locking them out of their essential data and systems. Spyware, as its name implies, covertly monitors user activities, collecting sensitive information such as login credentials, personal information, and financial details, which are then sent to the attacker.

Persistence techniques employed by malware are particularly sophisticated. These techniques ensure that malware remains active on a system despite reboots or attempts to remove it. Common methods include modifying system registries, utilizing rootkits to hide their presence, and exploiting legitimate software features to maintain control. Advanced malware can even update itself to evade detection by antivirus programs, adapt to new security measures, and exploit zero-day vulnerabilities—unknown flaws in software that have not yet been patched by the developer.

As the digital landscape evolves, so do the tactics of cybercriminals. Staying informed about the various forms of malware and their persistence techniques is essential for individuals and organizations to protect their digital assets effectively. Implementing robust cybersecurity measures, including regular software updates, comprehensive antivirus solutions, and user education, can significantly mitigate the risk posed by these malicious programs.

1.2. Problem Statement

The persistence of malware poses a significant threat to organizations and individuals alike. Despite advancements in cybersecurity, malware continues to evolve, utilizing sophisticated techniques to evade detection and maintain access to compromised systems. This persistence not only undermines the integrity of computer systems but also compromises data confidentiality and availability.

1.3. Objectives and Scope

The objective of this thesis is to analyze the persistence techniques employed by malware to evade detection and maintain persistence within compromised systems. The scope includes:

- Investigating historical trends in malware persistence techniques.
- Analyzing current methodologies in malware analysis, including static and dynamic analysis.
- Utilizing APIs such as VirusTotal and MalwareBazaar, as well as, other resources such as VirusShare and TheZoo repository on GitHub to gather and analyze malware samples.
- Mapping malware persistence techniques to the MITRE ATT&CK framework.
- Presenting findings through visual representations and analysis of predominant techniques and malware families.

1.4. Dissertation Structure

The Dissertation is divided into several chapters, where each chapter is devoted to some specific aspect of Malware persistence techniques and how they can be analyzed.

Chapter 1 (present chapter- Introduction, the author presents the general information related to the security of information in the digital age and pays attention to the importance of malware. It explains the kinds of malware including viruses, worms, Trojan, ransomware, and spyware, and stresses the need to address malware's persistence mechanisms to improve existing security features. The problem statement explains how viruses and malware threat remain a constant threat and how it has become increasingly sophisticated in how they cannot be detected easily as there is a need for better solutions. The goals and the concerns of the dissertation are stated with ideas about the historical and contemporary malware behavior persistence; their correlation to the MITRE ATT&CK framework; and detection strategies.

Next **Literature Review** analyses the existing information on malware as well as definitions explaining this type of threat and goes deeper into analyzing different types of malware and their abilities to persist. It even presents a timeline of the progression of malware and the main events that best define the further strategies used by malicious programs. The same is followed by current analysis approaches, static and dynamic, and a brief of how APIs like VirusTotal are used in malware analysis. Moreover, it discusses the role of malware in the development and escalation of cybersecurity and examines how these threats are addressed by the industry.

In the chapter of **Methodology,** we identify the samples of malware and organize and analyze them systematically, the present study has adopted a specific methodology as elucidated under Section 4, titled Methodology. To describe it – the research approach would be based on the combination of quantitative analysis of large sets of malware samples with the qualitative behavioral analysis of selected subsets. The focus of the chapter is on the identification of the source of malware samples, the scripts to automate the process, and data storage and organization. Future work will explore the integration between VirusTotal and other tools for further analysis. Additionally, this chapter presents considerations on the ethical concerns arising from the process of handling and analyzing malicious software.

In the chapter of **Case Studies / Analysis**, detailed descriptions of the structure of the used database for storing the data on various types of malware are provided, as well as the organization of tables and the kind of information stored. It also explains the CSV files that were created for the statistical analysis and follow-up visualization of the malware persistence methods. The chapter gives some information regarding the use of Python scripts in data collection, handling, and analysis, emphasizing on their significance in enhancing the research process. Furthermore, it looks into the outcomes of such analyses, which comprise the frequently utilized persistence practices, the effects on the identification rates, and the connection to the utilization of persistence and covering methods.

The **results** section provides the concluding point for the entire study, and it demolishes the most frequently used malware persistence techniques and their consequences. It uses heat maps and bar graphs to show how the persistence techniques are distributed by the malware's signature and its effect on detection. The chapter also looks at the correlation between persistence techniques and command and control techniques, and also the usage frequency of credential access techniques used by malware.

In the **discussion** chapter, major findings are analyzed, discussed, and related to the manipulative techniques generally employed in malware to evade detection. It examines the consequences of the majority of the persistence techniques for cybersecurity measures and the problems arising from multi-tactic malware. A discussion of the current detection technologies is also conducted to understand how well it functions, and the consequent further improvements that might be needed to counter the less frequent but more hazardous persistence types.

The chapter **conclusion and recommendations**, explores the challenges posed by malware persistence techniques, which enable malware to evade detection and persist in systems. It emphasizes the link between persistence methods and command and control strategies, highlighting the evolving tactics of malware developers. The study calls for an integrated approach to detection, improved security technologies, and continuous adaptation to combat both common and sophisticated threats. Collaboration and ongoing education are **recommended** to strengthen defenses against increasingly complex malware.

Finally, the **future works** chapter, outlines future directions for expanding research on malware persistence mechanisms. It suggests leveraging machine learning to improve predictive modeling, enabling the detection of emerging malware trends. Machine learning could also enhance the classification of malware by identifying subtle patterns, allowing for more precise categorization and targeted defenses based on industry-specific risks. The chapter emphasizes the potential of integrating threat intelligence platforms for real-time analytics and proposes cross-platform studies to cover a broader range of systems. It also recommends evaluating current defensive tools to identify gaps and improve malware detection strategies, highlighting the need for continuous advancements in cybersecurity research.

2. Literature Review

2.1. Definitions

Malware (Malicious Software): Malware refers to any type of software intentionally designed to cause damage, disrupt operations, steal sensitive information, or gain unauthorized access to computer systems. It encompasses a wide range of malicious programs, including viruses, worms, Trojans, ransomware, spyware, adware, and rootkits [1].

Malware Analysis

Analyzing malicious software, also known as malware, to get an understanding of its functioning, behavior, and effects on computer systems, is an important field in cybersecurity. Typically, through the analysis of malware, researchers attempt to comprehend the malware's behavior, objectives, and means of distribution, which is necessary for several reasons [2] [7]:

Virus: A self-replicating kind of malware extends by putting duplicates of itself into additional executable code or documents, a virus "Virus [3]." They frequently propagate through files that have been corrupted with infected email attachments or compromised websites. Several harmful effects can originate from this, ranging from data corruption to system instability. Virus It can spread among computer systems through any medium whether it is a diskette mail attachment or downloading software from untrustworthy sources like file sharing networks etcetera

Worm: A stand-alone malware program that copies itself and infects other computers or devices on the network is a worm "Worm." Unlike viruses, worms are designed in such a way as not to require a host program for them to spread themselves out. Worms take advantage of security holes to copy themselves across connected systems without any manual intervention [3].

Trojan (Trojan Horse): The malware that disguises itself under the name of an authentic application so that users are tricked into installing it, is "Trojan". Trojans are effective even when they do not exhibit signs of their existence, like installing additional malware, stealing crucial data disguising unauthorized access paths to the attackers' computers. Trojans are hidden in plain sight [4].

Ransomware: This is a type of malware that locks victim's files thus preventing their access from him/her, is ransomware [5]. Ransomware timing Attackers demand payment before rendering the files accessible through decryption as well as restoring access. There is widespread use of phishing tactics among ransomware attacks thereby exploiting human mistakes software errors etcetera

Spyware: A software program secretly monitoring a person's internet behavior without his/her awareness is spyware. It can record keystrokes track web-surfing patterns take down passwords as well and send sensitive information back to remote servers established by cybercriminals, such kind often operates inconspicuously to avoid getting noticed [6].

Adware: Malicious advertising on users' computers without their knowledge is known as adware. It often comes bundled together with legitimate software downloads, earning revenue through ad

impressions or clicks when ads are clicked [6]. Adware can reduce system performance, invade users' privacy, and also pave the way for additional malware infections

Rootkit/ Botnet: Rootkits are a particular kind of insidious malware that provides unauthorized access within a computer or network while avoiding detection from system administrators and security software [8].

Botnets refer to groups of computers linked together remotely without their owners' knowledge, mostly as part of operations such as spamming attacks, application of distributed denial of service (DDoS) methods, and financial frauds. APTs denote advanced acts of cyber aggression with clear objectives, initiated by professional hackers acting as proxies for their sponsoring institutions. The attacks carried out during such operations often result in the theft of sensitive information or disruption of business functions.

Recognition and Naming

Analyzing malware facilitates distinguishing between various types of malware including viruses, Trojans, worms, ransomware, and spyware. Through thorough scrutiny, experts can identify specific attributes and patterns that differentiate one type of malware from another.

Understanding Malware Capabilities

Looking at malware provides information on what capabilities it has and where those capacities might affect a system or network [2]. This involves knowing if it's able to steal data, sabotage operations, hijack resources or create backdoors that allow unauthorized access. This understanding is important when it comes to evaluating how dangerous each particular malware would be [7].

Propagation and Infection Methods

Malware often spreads using different methods of propagation such as emailing attachments, sneaky websites, USB drives, and network vulnerabilities. Therefore, analyzing them helps in knowing how they spread across different systems, thus helping make defense mechanisms stronger that can avoid future infections of this kind [7].

Mitigation and Remediation

Through malware analysis, one can design better approaches to contain the problem and find solutions (reduction strategies or elimination). Understanding how they work on computers leads developers to provide some kind of countermeasures aimed at eradicating parasites from affected computers and fixing any damaged system elements before identical problems reoccur in future instances.

Cyber Threat Intelligence (CTI)

Malware analysis is responsible for generating cyber threat intelligence that gives rise to certain attacks and malware trends. Such awareness enables companies to prevent unwanted situations in advance, focus on areas that need additional protection, and react fast in case there is a potential threat [9].

Forensic Investigation

During cyber incidents and data breaches, malware analysis is a crucial aspect of forensic inquiries. It helps link such events as trying to figure out when they started happening, where the first incursion was made by intruders, or what kind of data they tried to steal among others.

Security Tool Development

The functionality of various security tools has been informed by the lessons learned from examining malware instances [10]. These consist of antivirus programs; endpoint protection solutions such as intrusion detection system (IDS) software packages or threat intelligence systems that utilize techniques of analyzing malware for purposes like enhancing detection ability or preventing them.

Researching harmful software applications provides essential knowledge and instruments required by cyber experts in their ongoing fight against ever-changing threats related to computers. Therefore, businesses should learn how to detect these programs through scrutinizing them so that they can protect their information well since they are the most susceptible to such risks.

2.2. Historical Context

Since its inception, malware has undergone great changes as a result of alterations in computing technologies, trends in cybersecurity, as well as motives of people with ill intentions. Knowing where the malware originated gives us a perspective on how it has grown over time, its effects on digital systems, and strategies used to fight it [11].

2.2.1. Initial Development of Malware

Before the internet, malware history can be traced back to the early days of computer technology, when malicious software began coming up alongside new digital connections [12] between computers and other devices for individual use. Initial malware included experimental programs showing how code could disrupt or interfere with systems [2].

Malware has caused extensive disruption to digital systems, ranging from minor inconveniences to significant financial and operational damage. High-profile attacks, such as the WannaCry ransomware attack in 2017, demonstrated the potential for malware to disrupt critical infrastructure, including healthcare, transportation, and government services. The impact of such attacks is not limited to immediate financial losses but also includes long-term reputational damage and loss of trust.

One of the most significant impacts of modern malware is the theft of sensitive data. Advanced malware can infiltrate systems undetected, exfiltrating valuable information such as intellectual property, personal identification information, and financial records. Data breaches resulting from malware attacks can have severe consequences for individuals and organizations, including identity theft, financial fraud, and regulatory penalties.

2.2.2. Malware History Milestones

Emergence of Viruses 1980s: The 1980s saw the establishment of computer viruses, of which brain virus in 1986 is the most infamous one, marking a significant point in the history of malware. They were capable of replicating themselves and spreading from one computer to another [12].

Expansion of Malware Types, 1990s: The 1990s was a period when variety and technicality increased in varieties of malware. During this decade, there were many worms such as the Morris Worm dated 1988 whose purpose was to exploit weaknesses to move from one interconnected network to another [11][12].

Rise of Trojans and Botnets in the mid-2000s: Transitioning into the mid-2000s marked a change towards the domination of Trojans and botnets as essential forms of malware. Trojans used to be disguised as valid applications to deceive users into performing wrong actions with them, while Botnets employed infected devices to carry out large-scale attacks, especially distributed denial of services (DDoS) attacks.

2.2.3. Evolution of Techniques Employed by Malware

Stealth, Persistence

Malware progressed to techniques that would help them remain undetected and continue to execute themselves even on compromised systems [14]. To hide its existence and resist eradication, we had methods like rootkit installation as well as polymorphism

Rootkit Installation

Rootkits are a type of malware designed to gain unauthorized root or administrative access to a computer system. Once installed, rootkits can hide their presence by intercepting and modifying operating system calls, making them invisible to traditional detection methods such as antivirus software. Rootkits can conceal files, processes, network connections, and even other malware, creating a hidden environment within the host system. This ability to operate covertly allows attackers to maintain long-term access, gather sensitive data, or use the compromised system as a platform for further attacks without detection.

Rootkits can be classified into different types based on their operating level:

- **User-Mode Rootkits**
- **Kernel-Mode Rootkits**
- **Bootkits**

User-Mode Rootkits

Two types of rootkits work on the application layer; at user mode, rootkits operate in a way that might be perceived as intercepting and editing system calls executed by normal service requests. This mode

helps you to customize the behavior of these programs without changing the mainframe's OS itself. Through this process, user-mode rootkits effectively conceal themselves and other harmful actions (against which standard system calls & behaviors are relied upon for identification) from discovery [22].

One of how they go about this is hooking, which means redirecting system calls not to benign functions but to malicious ones throughout the operation. For example, as a case study, if someone orders for instance AV software, it will show a list of all running programs present in the system provided by Microsoft's Task Manager processes window. This enables the kit as well as other associated malicious software to remain invisible through routine monitoring tools. For example, user-mode rootkits also can alter the results of file system searches so that they do not include harmful data or present it as something else.

Unlike kernel-mode rootkits, which are programmed to remain undetectable at all costs, despite being hacked, user-mode rootkits have limitations. They can be easily detected and removed through modern computer protection devices working on top-bottom stacks, similar to theirs, too. When we talk about this, a case in point is security software at the kernel level bypassing user hooks and accessing data at the hardware level, thereby exposing the covert acts of the rootkit. Rather than that, the operational continuity of these kinds of rootkits may be interrupted when the operating system or the targeted applications undergo an update, making them not as robust as their kernel-mode analogs would be. All the same, user-mode rootkits remain serious threats because they are good at keeping their sneaky activities secret without melding into the system.

Kernel-Mode Rootkits

Kernel-mode rootkits operate at the kernel level of an operating system, providing them with extensive control over the entire system. By integrating themselves into the kernel, these rootkits gain access to critical system components and functions, allowing them to hide more effectively and perform a wide range of malicious activities with greater stealth and persistence. This deep integration makes kernel-mode rootkits particularly dangerous and challenging to detect and remove [24].

One of the key advantages of kernel-mode rootkits is their ability to intercept and modify low-level system calls, giving them the power to alter fundamental operating system behaviors. They can manipulate core system processes, file systems, and hardware interfaces, effectively concealing their presence and the presence of other malware. For example, a kernel-mode rootkit can intercept disk read/write operations, hiding its files and those of other malware from both the operating system and security software. It can also alter network traffic, making it possible to covertly exfiltrate data or communicate with command-and-control servers without detection [24].

Due to their deep integration into the operating system, kernel-mode rootkits are significantly more difficult to detect and remove compared to user-mode rootkits. Traditional antivirus and security tools, which operate at the user level, often cannot detect the rootkit's activities because these tools rely on the very system calls and functions that the rootkit controls and manipulates. Advanced detection methods, such as kernel integrity checkers and specialized rootkit scanners, are required to identify anomalies at the kernel level. However, even these tools can struggle to keep up with the evolving techniques used by kernel-mode rootkits, such as direct kernel object manipulation (DKOM) and virtual machine-based rootkits (VMBRs), which further complicate detection efforts.

Bootkits

Bootkits are a unique type of malicious code that alters the startup process of an operating system, so they can be activated before the system's security features come into operation. Therefore, to enable themselves to continue working during system boot time, bootkits isolate themselves in their early phases by taking control of it. As a result, long before other protective measures become operational, their harmful software gets executed. This makes bootkits invisible because they will always have some time margins before any reactions are noticed [25].

One popular technique applied by bootkits includes modifying the Master Boot Record (MBR), Volume Boot Record (VBR), or Unified Extensible Firmware Interface (UEFI). With these components changed, bootkits make sure their malicious codes are foremost during system start-up time. This way, they can manipulate the order and manner of loading the OS to avoid detection alongside other possible malware. Such a level of mastery while starting a Personal Computer would entail utilizing specialized software meant for noticing such inconsistencies at an early stage of the boot process when these security tools are activated only in the latter stage.

A good example is how bootkits are usually designed them withstand reboots even in cases where you reinstall your OS numerous times over again because these types of code can embed themselves into certain areas that remain untouched whenever one performs a normal operating system reinstallation For instance, it could be inside system's firmware or take advantage of obscure storage locations found within its hard disk which OS never overwrites in the usual way Another thing worth noting here is that having such kind of durability against elimination makes them extremely complicated when it comes down to methods for removal as you would need specialized programs capable of completely purging out infection from boot sectors as well as firmware which has been contaminated [25].

Polymorphism

Polymorphic malware takes stealth to another level by continuously changing its code to evade detection. Polymorphism involves using algorithms to encrypt the malware's code, which is then decrypted and executed at runtime. Each time the malware replicates or infects a new system, it generates a new, unique version of its code, making it challenging for signature-based detection methods to identify it.

Key characteristics of polymorphic malware include:

Encryption and Decryption

Encryption and decryption are sophisticated techniques used by malware to protect its core payload and evade detection. In this process, the malware encrypts its malicious code using a unique key for each infection, making it difficult for traditional security measures to recognize and intercept it. This method of dynamic encryption ensures that even if one instance of the malware is detected and analyzed, other instances remain concealed due to their unique encryption keys [20].

The encryption process involves encoding the core payload of the malware in such a way that it cannot be easily understood or executed by the system without the correct decryption key. When the malware

is executed on a target system, a small decryption routine embedded within the malware is triggered. This routine uses the specific key associated with that infection to decrypt the payload, allowing the malicious code to execute and carry out its intended functions. By hiding the true nature of the payload until the moment of execution, the malware can avoid detection by static analysis tools that scan files for known signatures [26].

This technique of using unique encryption keys for each infection, often referred to as polymorphism, significantly enhances the malware's ability to evade detection. Each encrypted payload appears different to security software, which relies on pattern recognition to identify threats. Polymorphic malware continuously changes its appearance, making it challenging for signature-based detection methods to keep up. As a result, even if the decryption routine is identified and flagged, the constantly changing payload encryption ensures that each new instance of the malware requires a fresh analysis.

The use of encryption and decryption in malware is not only a means of obfuscation but also a method to protect the integrity of the malicious code against reverse engineering. Security researchers attempting to analyze the malware's behavior face the additional hurdle of decrypting the payload, often requiring significant time and computational resources. This delay allows the malware to operate longer on infected systems before being neutralized, increasing its potential impact [26].

Mutation Engines

Mutation engines are used in polymorphic malware to modify the code’s appearance while preserving its fundamental purpose. This continuous mutation process leads to each instance looking distinct, thus making its identification difficult by usual protection means.

The process of mutation makes use of numerous strategies to change the code for malware. One common way is to put nonsense instructions in the code that do not affect its behavior but will alter its binary fingerprints, such as the insertion of random garbage in the instruction stream or meaningless arithmetic computation. No-ops may be used as an example when padding.

Mutation engines can also reorganize sequences of code different from those that were before, changing how operations were carried out or replaced with similar commands. For instance, some apps may reorder their function calls so that others come first, while some can use diverse commands that perform the same tasks other than those already existing. It makes the new malware look different from the previous one, leading to its masking [27].

Mutation engines may also change how a virus’s payload is coded. When the code sequence and method employed for encryption are altered including keys used to encrypt data during self-replication processes, every subsequent replication will present differently encrypted content of the malware. This behavior makes sure that even if someone manages to crack one version of cryptography used in this file, the following variant will continue being protected by another encryption [27].

Their capability to spawn an almost inexhaustible number of distinctive replicas from one single piece of malware is what renders mutation engines so effective. This can quickly overwhelm traditional antivirus software that relies on known signatures or patterns for detection purposes, hence newly mutated ones evade detection even if one version has been identified and included in the signature database [27].

Dynamic Code Generation

Dynamic code generation is an advanced technique used by sophisticated polymorphic malware to further complicate detection and analysis. This method involves the malware creating new, functional code segments on the fly during its execution, ensuring that each instance of the malware is unique and behaves slightly differently [28].

The process of dynamic code generation allows the malware to adapt its code in real-time, making it extremely difficult for traditional security solutions to identify and counteract it. Unlike simple polymorphism, which primarily alters the appearance of the code, dynamic code generation can modify the actual functionality of the malware. This includes creating new algorithms, altering execution paths, and introducing variations in the way tasks are performed, all while maintaining the malware's malicious objectives.

One of the primary benefits of dynamic code generation for malware developers is the ability to evade signature-based detection methods. As the malware generates new code segments during execution, it presents a constantly changing footprint that signature-based antivirus programs cannot easily track. Each time the malware runs, it produces a different set of instructions and behaviors, making it nearly impossible for static analysis tools to detect a consistent pattern [28].

Moreover, dynamic code generation can also help the malware adapt to different environments and evade behavioral analysis. By changing its code and execution strategy based on the environment it finds itself in, such as the presence of specific security tools or the type of operating system, the malware can tailor its behavior to avoid detection. For example, it might generate benign code segments when it detects a virtual machine or sandbox environment, a common tactic used by security researchers to analyze malware [28].

Advanced Persistence Techniques

Beyond rootkits and polymorphism, modern malware employs various other persistence techniques to ensure continued operation and resistance to removal.

Fileless Malware

Fileless malware represents a sophisticated type of cyber threat that operates directly in a computer's memory (RAM) without leaving any traces on the hard drive. This method of operation significantly complicates detection and removal efforts because traditional file-based antivirus programs, which scan for malicious files stored on disk, are unable to identify such threats [29].

The primary advantage of fileless malware is its ability to execute malicious activities without writing any code to the disk. This stealthy approach allows the malware to evade detection mechanisms that rely on identifying suspicious files and signature-based detection methods. Instead, fileless malware often leverages legitimate system tools and processes to carry out its malicious actions, blending seamlessly into normal system operations [30].

Fileless malware typically infiltrates a system through techniques such as exploiting vulnerabilities in web browsers, email attachments, or malicious links. Once inside, it can inject malicious code into running processes or use tools like PowerShell, Windows Management Instrumentation (WMI), or other scripting environments to execute its payload. By utilizing these legitimate tools, the malware can perform various tasks, such as data exfiltration, credential theft, and system manipulation, all while remaining largely undetected.

The persistence of fileless malware is another significant challenge. Because it resides in memory, the malware can be eliminated by simply rebooting the infected system. However, sophisticated variants can establish persistence by modifying system configurations, such as creating registry entries or scheduling tasks that re-inject the malicious code into memory upon system startup. This ensures that the malware can survive reboots and continue its malicious activities [29][30].

To counter fileless malware, cybersecurity strategies must evolve beyond traditional file-based detection methods. Advanced techniques, such as behavioral analysis, heuristic-based detection, and memory forensics, are crucial in identifying and mitigating fileless threats. Behavioral analysis monitors system activities in real-time, looking for anomalies and suspicious behaviors indicative of malware. Heuristic-based detection involves analyzing the behavior and characteristics of processes to identify potential threats based on patterns rather than specific signatures.

Persistence Mechanisms Utilizes legitimate system features such as scheduled tasks, registry keys, or system services to reinitialize the malware upon system reboot or at specific intervals.

Command and Control (C&C) Servers

Command and Control (C&C) servers are integral components of sophisticated malware operations, facilitating communication between the malware and remote servers controlled by attackers. These servers enable attackers to maintain control over infected systems and receive instructions, updates, and additional payloads, thereby allowing them to adapt their tactics based on the security environment of the compromised network.

C&C servers provide a robust infrastructure for managing large-scale malware campaigns. Once a system is infected, the malware establishes a connection with the C&C server, often using encrypted communication channels to avoid detection by security measures. This connection allows the attackers to send commands to the malware, directing it to perform specific actions such as data exfiltration, system reconnaissance, or the deployment of additional malware. The ability to issue real-time commands enables attackers to dynamically alter their strategies in response to the defenses they encounter.

One of the primary advantages of C&C servers is their ability to deliver updates and new payloads to the malware [31].

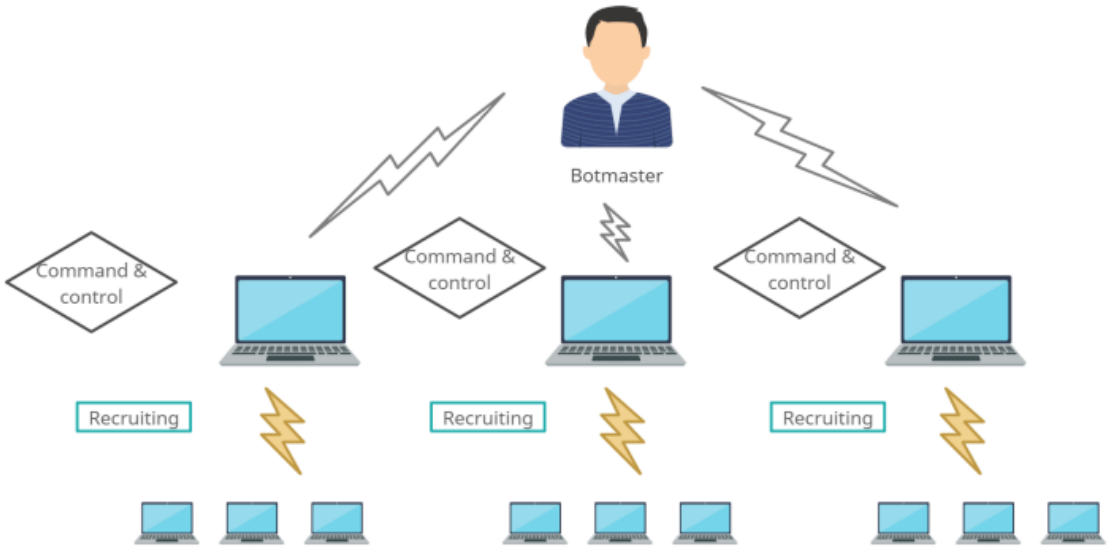

Figure 1 - Command and control mechanism [32]

This capability allows the malware to evolve and adapt, incorporating new functionalities or exploiting newly discovered vulnerabilities. For example, if a security patch is released to counter the initial malware infection, the C&C server can push an update to the malware, enabling it to bypass the new defenses. This dynamic adaptability makes C&C server-based malware highly resilient and difficult to eradicate.

C&C servers also facilitate coordination and synchronization among multiple infected systems. In large botnets, where thousands of compromised devices are under the control of a single attacker, the C&C server acts as the central command hub. It orchestrates distributed denial-of-service (DDoS) attacks, data theft operations, or spam campaigns by coordinating the actions of individual bots. This centralized control allows attackers to execute complex, large-scale operations with precision [32].

To counter the threat posed by C&C servers, cybersecurity professionals employ various detection and mitigation techniques. Network monitoring tools can identify unusual patterns of communication that may indicate the presence of a C&C connection. Intrusion detection systems (IDS) and intrusion prevention systems (IPS) can analyze network traffic for known indicators of compromise associated with C&C activity. Additionally, threat intelligence services provide information on known C&C server IP addresses and domains, enabling organizations to block or monitor connections to these malicious endpoints.

Advanced mitigation strategies include sinkholing and takedown operations. Sinkholing involves redirecting traffic destined for a C&C server to a controlled server operated by security researchers or law enforcement. This disrupts the communication between the malware and its controllers, effectively neutralizing the threat. Takedown operations target the infrastructure of the C&C servers themselves, often in coordination with international law enforcement agencies, to dismantle the command network and arrest the operators [32].

2.2.4. Countermeasures and Detection

To combat these sophisticated stealth and persistence techniques, cybersecurity professionals have developed advanced detection and mitigation strategies:

Behavioral Analysis

Observing the behavior of programs in a controlled environment (sandboxing) to detect suspicious activities that indicate the presence of malware.

Heuristic Analysis

Analyzing code for characteristics typical of malicious behavior allows the detection of new and unknown malware variants.

Memory Forensics

Examining the memory of running systems to identify anomalies and hidden processes indicative of rootkits or fileless malware.

Endpoint Detection and Response (EDR)

Continuously monitoring endpoints for unusual behavior, enabling rapid detection and response to threats.

Financial Motivation

The shift from amateurish hackers into criminals who were driven by money led to development followed by high-level sophistication in malware. This prompted the creators to focus their efforts on stealing financial data from unsuspecting individuals instead of conducting frauds, among other things, he used to make money by taking advantage of software weak points [13].

Advanced Persistent Threats (APTs): APTs emerged during the millennium, with this being characterized as a time when highly adept digital campaigns were organized mostly by either governments or even groups of criminals specializing in online crime. The suspects in question engage in espionage, steal intellectual property, or disrupt strategically targeted organizations [14].

Defensive Technologies

Over the years, the cybersecurity industry has developed a range of defensive technologies to combat malware. Antivirus software, firewalls, intrusion detection systems (IDS), and intrusion prevention systems (IPS) are fundamental tools in the fight against malware. These technologies have evolved to incorporate advanced techniques such as heuristic analysis, machine learning, and behavior-based detection to identify and mitigate threats.

Patch Management and Vulnerability Assessment

Regular software updates and patch management are critical in protecting systems against malware. Vulnerability assessments and penetration testing help identify and address security weaknesses before they can be exploited by attackers. Organizations must maintain an up-to-date inventory of all software and hardware assets to ensure the timely application of patches and updates.

Education and Awareness

Human factors often play a significant role in the success of malware attacks. Phishing remains one of the primary methods for delivering malware. Educating users about safe computing practices, recognizing phishing attempts, and the importance of strong, unique passwords can significantly reduce the risk of malware infection. Security awareness training should be an ongoing effort, reinforced with regular simulations and updates on emerging threats.

Incident Response and Recovery

Despite the best preventive measures, incidents can still occur. Having a robust incident response plan in place is essential for mitigating the impact of a malware attack. This plan should include procedures for detecting and containing the malware, eradicating the threat, and recovering affected systems. Regular backups and a well-defined disaster recovery plan ensure that organizations can restore operations quickly and minimize downtime.

2.2.5. Effects on the Cybersecurity Landscape

The transformative impact of malware on cybersecurity practices includes:

- Security Industry Response: The rise of malware saw the development of antivirus software, intrusion detection systems (IDS), and malware analysis tools for threat detection, analysis, and mitigation [15].
- Regulatory and Legal Responses: Governments passed regulations and laws that were aimed at combating cybercrime and safeguarding digital infrastructure.
- Cybersecurity Awareness: The spread of malware highlighted the need for individual users as well as enterprises and organizations to be aware of network threats.

2.2.6. Current Trends and Future Outlook

Currently, malware is adapting to new technologies such as artificial intelligence (AI) and the Internet of Things (IoT). Some future challenges include how to prevent AI-drive attacks and secure IoT devices, as well as filling the global cybersecurity skills gap.

2.3. Review of Current Techniques

2.3.1. Static Analysis

Static analysis is a crucial method in the cybersecurity toolkit, aimed at examining malware without executing it in a live environment. This approach involves a meticulous inspection of the malware's code and structure to identify characteristics and behaviors indicative of malicious intent. By analyzing the source codes or executable files, security professionals can uncover telltale signs of malware, such as specific API calls, encrypted code segments, and hard-coded URLs that facilitate communication with Command and Control (C&C) servers [33].

One of the fundamental steps in static analysis is checking file headers. File headers contain metadata about the file, including information about its format, creation date, and author. By examining these headers, analysts can gain insights into the malware's origin and potential modifications, which may reveal attempts to disguise its true nature.

Analyzing strings contained within an executable file is another key aspect of static analysis. Strings are sequences of readable text embedded in the code, often revealing valuable information such as command-line arguments, error messages, and URLs. Extracting and scrutinizing these strings can help identify the malware's intended actions, communication channels, and targeted assets. For instance, hard-coded URLs might indicate the addresses of C&C servers or sites used for data exfiltration [34].

Disassembling or decompiling code is a more in-depth approach to static analysis. Disassembling involves converting the executable code back into assembly language, a human-readable format that reveals the low-level instructions executed by the CPU. Decompiling goes a step further by attempting to reconstruct the source code from the compiled binary. These processes allow analysts to understand the malware's logic, functionality, and potential impact. By examining the code structure and flow, they can identify harmful behaviors such as file deletion, data encryption, or unauthorized network access [33].

Additionally, static analysis includes identifying and analyzing specific API calls made by the malware. API calls are functions provided by the operating system or other software libraries that the malware uses to perform various tasks. Malicious software often relies on particular APIs to execute its payload, manipulate files, or communicate over the network. Recognizing these calls can help pinpoint the malware's capabilities and potential damage.

Encryption is another aspect often scrutinized during static analysis. Many sophisticated malware samples encrypt portions of their code to evade detection. Analysts must identify these encrypted segments and attempt to decrypt them to fully understand the malware's functionality. This process often involves reverse engineering the encryption algorithm used by the malware.

Moreover, static analysis can uncover embedded obfuscation techniques designed to hide the malware's true purpose. Obfuscation may involve renaming variables and functions to meaningless names, inserting redundant code, or using complex control flows. Analysts use de-obfuscation tools and techniques to simplify the code and reveal its underlying logic.

While static analysis provides a wealth of information, it has limitations. It cannot reveal the dynamic behavior of the malware when executed, such as how it interacts with the system, network, or user inputs. Therefore, static analysis is often complemented by dynamic analysis, where the malware is run in a controlled environment to observe its real-time action [16].

2.3.2. Dynamic Analysis

On the other hand, dynamic analysis involves executing the malware under confined conditions like in a sandbox or on virtual machines so to see what happens during its run time. This helps capture behavior like interactions with the operating system, network communication patterns, modifications made to the file system, or any memory consumed by it. Monitoring system calls made by the application, device activity within a network or changes in systems state enables understanding the dynamic behavior of a malware that facilitates understanding of its functionalities and motivation behind creation [17].

Dynamic analysis typically takes place in confined conditions such as sandboxes or virtual machines. A sandbox is an isolated environment specifically designed to safely execute and analyze potentially malicious code without risking harm to the actual system. Similarly, virtual machines provide a contained setting that mimics real-world computing environments, allowing the malware to operate as it would on a physical machine [35].

During dynamic analysis, several key behaviors are closely monitored. One crucial aspect is the malware's interaction with the operating system. This includes observing system calls made by the malware, which can reveal attempts to access or manipulate system resources. For instance, the malware might try to open files, create processes, or modify system settings. By tracking these calls, analysts can identify the specific actions the malware performs to achieve its objectives [35].

Network communication patterns are another critical element of dynamic analysis. Malware often needs to communicate with external servers to receive commands, exfiltrate data, or download additional payloads. By analyzing network traffic generated by the malware, security professionals can identify the destinations of these communications, the nature of the data being transmitted, and any protocols used. This information is essential for understanding the broader network of compromised systems and the attackers' command and control infrastructure.

Dynamic analysis also focuses on changes made to the file system and memory during the malware's execution. Malware may create, delete, or alter files and directories as part of its operation. It may also inject code into other running processes or allocate memory for malicious purposes. By monitoring these modifications, analysts can detect signs of data theft, file encryption (as seen in ransomware), or the establishment of persistence mechanisms that ensure the malware survives system reboots [36].

In addition to basic observations, dynamic analysis employs advanced monitoring techniques to capture more detailed information about the malware's behavior. This includes tracking device activity within a network, such as unusual patterns of data transfer or the activation of network devices that typically remain dormant. Changes in the system state, such as altered registry entries, modified configuration files, or unusual CPU and memory usage, are also closely examined to identify the full scope of the malware's impact.

The ultimate goal of dynamic analysis is to understand the malware's functionalities and the motivations behind its creation. By observing how the malware behaves in a real-world-like environment, analysts can deduce its primary objectives, whether they are financial gain, data theft, system disruption, or espionage. This knowledge is crucial for developing targeted defenses and mitigating the threat posed by the malware [36].

2.3.3. Behavioral Analysis

Behavioral analysis involves studying how malware acts while it executes itself. This means that it tries capturing abnormal activeness s indicated by activities like system settings alterations, data access requests to confidential information, or infecting other machines. Besides, behavioral analysis often gives additional insight into the operational methods of malware in compromised systems, since it complements both static and dynamic analysis [19].

2.3.4. Code Reversing

Code reversing is the process of scrutinizing either a malware’s binary code or its bytecode to reconstruct the forgotten source code [21]. To discover more secrets such as its control flow or encryption algorithm, one needs skills in assembly language programming, debugging, and reverse engineering tools. Usually, such is done to understand complex families of malware and come up with effective countermeasures against them.

2.3.5. Memory Analysis

Memory analysis entails scrutinizing the contents of the volatile memory (RAM) of a computer to reveal the active parts of malware and how it operates. This is fundamental for detecting stealthy malware that bypass traditional detection approaches through residence in only RAM or inserting code into lawful processes. They involve using memory forensics techniques to obtain process memory dump files to identify malware artifacts like hooks, injected code, or any process anomalies within them [22].

2.4. Recent Developments

The cybersecurity environment is dynamic since both the offense and the defense sides are changing through the use of fresh technologies and techniques. It has achieved this because of the shifting dynamic of the environment due to the enhanced nature of the threats posed by malware. Another prominent feature of this area is the application of such technologies as machine learning and artificial intelligence. These technologies have changed the prospects of threat detection and analysis by giving cybersecurity professionals the tool that helps discover the relationships and patterns that are hard to distinguish by conventional means. Machine learning portable programs which are learned from large sets of known malware can now predict and recognize another set, but unknown malware by looking at hints that may not be discernible by the human eye. Similarly, the current AI-based tools are also fairly flexible to evolve with new data being fed to the system to provide better detection advances over time.

New developments in sandboxing technologies in turn have added greater strength to dynamic analysis, which forms a part and parcel to study on how malware can work around its environment. Sandbox tools of the current generation are endowed with additional features that enable the functioning of

complex operating environments, enabling execution and analysis of malware effects on real systems in protected surroundings. All these technologies have become more effective in emulating different kinds of system configurations and user activities, which are crucial in eliciting all the possible behaviors of malware. Such detailed dynamic analysis has turned out to be completely useful in searching for stealthy or dormant components of viruses and other forms of malware that can remain inactive in more artificial conditions.

Besides the advances in sandboxing technology, threat intelligence platforms define how businesses gather, process, and counter threats in the domain of cybersecurity. Such platforms collect data from several distinct sources, such as threat feeds with worldwide coverage or even honeypots, as well as cybersecurity communities. Through putting up synthetic malware analysis channels, organizations can now evaluate large amounts of information within a short duration, thereby cutting down on the time that it takes to detect or contain new threat attacks. This automation not only enhances the speed as well as the effectiveness, but it also engages security teams in proactive, advanced threat hunting the ability to identify threats and prevent them from proliferating and inflicting substantial harm to the network.

Also, the emergence of cloud-based security solutions has led to the appearance of more scalable and flexible approaches to malware analysis. Several operation platforms may provide substantial computational resources, which are important for the execution of strenuous machine-learning algorithms and big-scale sandboxing settings. Here, the transition to the cloud has made the more complex analysis of malware within the reach of even the enterprise that might not have had the means to implement such functionality before.

These developments reflect a growing trend towards automation and real-time analysis in the fight against increasingly complex and evasive malware. As cyber threats continue to evolve, so too must the tools and techniques used to combat them. The integration of AI and machine learning, along with advancements in sandboxing, threat intelligence, and cloud-based solutions, represents a significant step forward in enhancing the overall effectiveness of cybersecurity defenses. However, these advancements also underscore the ongoing arms race between attackers and defenders, highlighting the need for continuous innovation and adaptation in the cybersecurity field.

2.5. Knowledge Gaps

There are, also, several knowledge gaps that, although we have in recent years seen some advancement in the approach to dealing with them, persist as major issues for cybersecurity professionals. As one of the identified issues, this is particularly a significant weakness given that fileless malware has become one of the biggest threats to organizations' security. Whereas conventional malware always includes executable files that are stored on a disk or a file, fileless malware does not require writing anything to the disk; they run their code from the RAM alone. This characteristic makes it extremely hard to identify and mitigate such threats using conventional analysis tools that work through file signatures and disk-based artifacts. Hiding inside the computer memory, fileless malware has recently become an increasingly popular tool in the hands of cybercriminals and as such there is a clear need to develop sophisticated tools capable of real-time monitoring and analysis of the in-memory processes to effectively counter this threat.

One of the areas that need more exploration is the identification and analysis of APTs, Advanced Persistent Threats. APTs are complex, protracted attacks that may be executed by a state actor, criminal organization, or other advanced cyber threat agents. These attacks are gradual, and attackers may take months, years, or longer to complete their intrusive operations in a targeted firm's network. APTs are complex and the modern ones use polymorphic techniques, and encryption and mimic the official processes of the system, thus being able to evade simple antivirus scans. These APT techniques complicate matters, hence are a nightmare for even advanced security teams, given that APTs are more sophisticated than most other hacking types. Hence, there is a dire need for advanced and effective tools that can not only identify the first phase of intrusion but also monitor the further activities of these threats, so that the concerned organizations get adequate time to counteract and reduce their losses.

To that, we can add that polymorphic and metamorphic malware analysis have not received sufficient attention and present one of the significant gaps in the field. These are the malware programs that build up new code in the infected machine with each repetition, and this makes them very hard to detect through conventional methods of scanning for signatures. Polymorphic malware resamples a copy of the original code with a different code, yet it serves the same function as the preceding one, while metamorphic malware replays the program at a different level with a completely new set of code, making it hard to detect. It is due to the high evolution rate of the malware types in question that even the currently active analysis techniques cannot efficiently prevent and contain the corresponding threats, thus resulting in escalations of undiagnosed infections. In the absence of these, there is a necessity for the emergence of sophisticated heuristic and behavior-based analysis tools to identify these threats from their behavior patterns rather than on the structure of their code. In addition, the building of recognizers as capable of recognizing patterns in a Family of malware might be a more proactive way of detecting and preventing the unyielding threats.

In addition, the unfolding developments related to the significance of the Internet of Things (IoT) devices that have found their way into society is another research gap. IoT devices are generally described as less powerful devices with more restricted security measures and have become cyberattack favorites. Because the IoT security field is not well-defined especially in the aspect of security, with more connected and different breeds of devices the process of defining strategies to contain malware in IoT becomes a tough nut to crack. Most of the present-day malware identification methods cease to be effective in mitigating the threats of continually emerging IoT systems, where measures such as antivirus cannot be applied. Hence, findings on smaller and more easily deployable forms of security that can be adopted across various IoT contexts are important to fill this gap of research.

Addressing these knowledge gaps is crucial for developing more robust cybersecurity defenses and staying ahead of the rapidly evolving threat landscape. As attackers continue to refine their methods, the cybersecurity community must prioritize research and innovation to overcome these challenges, ensuring that organizations are equipped to protect their critical assets against the most advanced and elusive threats.

3. Methodology

This chapter presents the methodical approach in this study to review malware's persistence mechanisms. The methodology falls under four main sections: Research Design, Data Collection, Analysis Methods, and Ethical Considerations. Every part shows how the study is all-inclusive, replicateable, and done ethically.

3.1. Research Design

Centered around a structured and methodical approach to collecting, categorizing, and analyzing malware samples with a specific focus on persistence techniques, the research design for this study is. The purpose of the study is to know how malware of different kinds remains in infected systems over a certain period.

Owing to the vast and constantly changing nature of the malware landscape, the research adopts a mixed-method approach. It combines quantitative analysis from examining large datasets of malware samples with qualitative insights resulting from in-depth behavioral analysis of selected samples. Therefore, its outcomes are both statistically significant and contextually meaningful due to the two-fold approach.

This study is divided into several stages which start with gathering reputable sources of malware samples, followed by holding this data within a secure database. The consecutive step of analysis involves categorization based on persistence techniques through automated means to extract detailed metadata and behavioral patterns. Lastly, this study also includes an elaborate consideration of the ethical issues related to handling and analyzing malicious software.

3.2. Data Collection

The data collection phase is a critical component of this research, as the quality and relevance of the data directly impact the validity of the findings. This section outlines the strategies employed to gather malware samples, ensuring that the data set is both comprehensive and relevant to the research objectives.

3.2.1. Sources of Malware Samples

The malware samples were collected from several well-established repositories and databases, including:

- MalwareBazaar [37]: a place with lots of different malware samples that can be easily reached through its API. For that purpose, numerous automated scripts were designed.
- VirusShare [38]: another significant site that has a large collection of malware. To ensure consistency and make it automated, the necessary hashes from these files were downloaded and extracted using scripts.

- TheZoo [39]: it is a GitHub repository that contains a wide range of malware samples. We cloned the entire repository and developed scripts that could identify SHA-256 hashes from the provided documents.

Initially, more than 5.5 million malware hashes were gathered. This amount was trimmed down to 22,366 after extensive sieving for duplicates and unrelated examples. VirusTotal's API call restrictions were the main reasons why the dataset was further refined to 18,296 samples from July 2023 up-to-date having known signatures only to reduce the scope of our analysis. This step was necessary as it would make our analysis process easier, and we could get accurate information based on statistics.

3.2.2. Script Development for Automation

To handle the large scale of data collection, Python scripts have been developed for automating the process of collecting and analyzing malware samples. These scripts have been performing the following duties:

- Downloads and Hash Extraction: Designed to take malware samples from the cited sources above, these scripts can download malware samples from the previously mentioned sources and then get only those hashes that are essential for further analysis.
- Error Handling and Retries: Scripts were equipped with strategies to handle errors nicely. This included retrying failed downloads or API calls to avoid overloading servers and making sure data collection is reliable.
- Progress Logging: Detailed logging was used to keep track of how far data had been collected, which made monitoring and troubleshooting easy.

Through automation, not only was human time saved but humanly erroneous occurrences were also minimized, hence consistency for reliable data.

3.3. Data Storage and Management

With the malware samples collected, the next step was to store and manage the data effectively. Given the sensitivity and volume of the data, a robust database system was required.

3.3.1. Database Setup

A MySQL database was chosen for this research due to its scalability and ability to handle large datasets efficiently. The database schema was carefully designed to accommodate the various types of data collected, including:

- **all_hashes**: This table stores all the collected hashes.
- **vt_hashes**: A subset of hashes for which detailed metadata from VirusTotal is available.
- **other_hashes**: Contains hashes obtained from other sources not covered by the first two categories.
- **malware_info**: This table stores metadata related to each hash, such as the date of first appearance and the associated persistence techniques.

These tables are interlinked to allow for efficient querying and analysis, facilitating the extraction of meaningful insights from the data.

3.3.2. Data Insertion and De-duplication

After the database had been established, the team created scripts to input data automatically. These scrips checked existing entries against hashes of inbound entries so they can be disposed in case they contain duplicates. The purpose of this activity was to prevent distorting analysis results using superfluous information while still preserving its coherence and completeness.

3.4. Analysis Methods

The analysis phase focused on categorizing the malware samples based on their persistence techniques and extracting meaningful insights from the data. Automated tools played a significant role in this process, ensuring that the analysis was both thorough and efficient.

3.4.1. Integration with VirusTotal

One of the central tools utilized in this research was VirusTotal [40], a widely recognized online platform that aggregates data from over 70 antivirus engines and various other security tools. VirusTotal plays a pivotal role in enhancing the understanding of malware by providing a comprehensive overview of how different security vendors classify and respond to particular malware samples. The integration of VirusTotal into this research was instrumental in collecting detailed and reliable metadata for each malware sample, contributing significantly to the overall analysis.

The integration with VirusTotal provided a substantial foundation for the research, allowing for a more detailed and comprehensive analysis of malware persistence techniques. The wealth of metadata and behavioral insights obtained from VirusTotal enriched the research database, making it possible to draw more accurate and meaningful conclusions. Moreover, the ability to validate and cross-reference data from multiple sources added a layer of reliability to the findings, ensuring that the research outcomes are both credible and robust.

Figure 2 - Virus Total tool

A Python script was developed to send requests to VirusTotal for each hash, retrieving detailed metadata, including SHA-256 hashes, file signatures, and behavioral reports. This metadata was then cross-referenced with the data stored in the database to ensure accuracy and completeness.

To effectively leverage VirusTotal's capabilities, a Python script was meticulously developed to automate the process of interacting with the VirusTotal API. This script systematically sent requests for each malware hash stored in the database, querying VirusTotal for a wealth of information. The data retrieved for each hash included:

SHA-256 Hashes: These serve as unique identifiers for the malware files, ensuring that the correct sample is being analyzed and reducing the chances of duplication in the dataset. The script checked the retrieved SHA-256 hashes against those in the database to confirm that the correct data was associated with each malware sample.

File Signatures: VirusTotal aggregates signature data from numerous antivirus vendors, offering insights into how different engines identify the malware. This includes the specific malware family or type as identified by various vendors, which is crucial for understanding how widespread and recognized a particular threat is. The script compared the file signatures from VirusTotal with any previously collected signature data, highlighting any discrepancies that might indicate a misclassification or an update in the malware's identification by antivirus vendors.

Behavioral Reports: These reports provide in-depth details about how the malware behaves when executed in a controlled environment. They include information on file system changes, network activity, registry modifications, and any persistence mechanisms that the malware may employ. This

behavioral data is essential for identifying the specific techniques malware uses to remain active on infected systems. The behavioral reports from VirusTotal were cross-referenced with other data sources to ensure consistency. For example, if a particular persistence technique was identified in VirusTotal's behavioral analysis, the script checked for similar findings in other sandbox environments or analysis tools. This step was crucial for validating the accuracy of the persistence techniques documented in the research.

3.4.2. Analysis Using MalwareBazaar API

In addition to VirusTotal, the MalwareBazaar API was utilized to retrieve detailed information about each hash. This included the date of first appearance, the malware family it belongs to, and its classification. The data from MalwareBazaar was instrumental in identifying trends and commonalities among the samples, particularly in terms of persistence techniques.

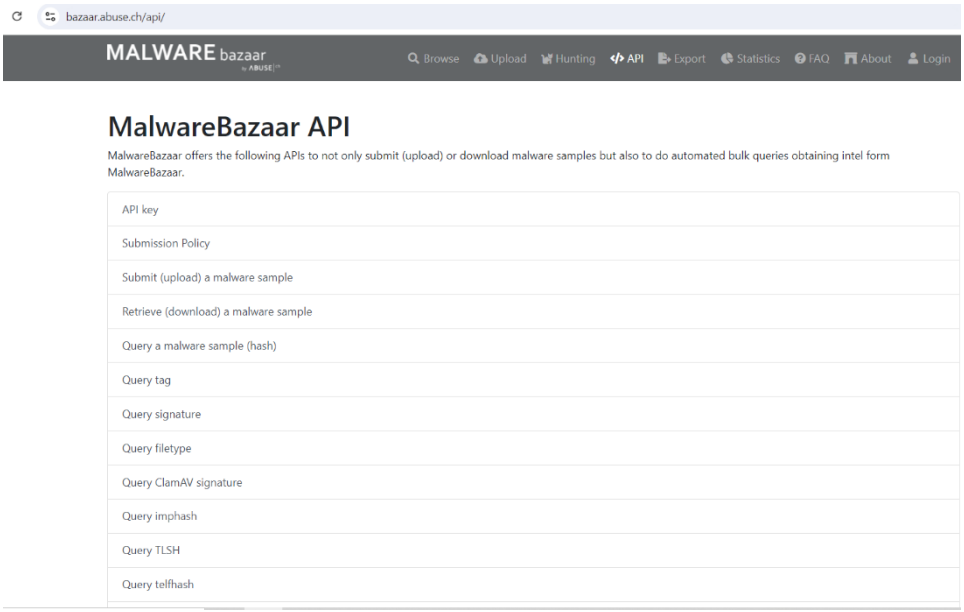

Figure 3 - MalwareBazaar API

3.4.3. De-duplication and Filtering

To enhance the relevance of the analysis, a de-duplication process was conducted, merging hashes from different sources and removing any redundancies. The dataset was then filtered based on the date of first appearance, focusing on recent malware samples (post-July 2023). This filtering ensured that the analysis was up-to-date and reflective of the current threat landscape.

3.4.4. Detailed Behavioral Analysis

Selected samples were subjected to more in-depth behavioral analysis using online sandboxes such as VirusTotal. These sandboxes provided detailed reports on how each sample behaves in a controlled environment, with a particular emphasis on persistence techniques. The results were then added to the database, linking each hash with its corresponding behavioral data.

One of the crucial roles played by the data obtained from VirusTotal was to validate the collected malware samples and act as a foundational element for further in-depth analysis of the entire malware dataset. By categorizing malware samples, analysts were capable of having an idea of what some specific ones could be used for amongst others. The classification information available at VirusTotal helped in organizing it by its originators into different groups or families. This particular kind of organization had to do with observing some similarities between different types of worms over time thereby uncovering how they were related as well as their behavioral traits respectively. However, it provided a basic guide for exploring various persistence techniques later on because certain malware groups are known to apply exact strategies in order not to leave infected machines. Hence, this investigation would pay much attention to how each family maintains itself on a computer or network across a cybercrime campaign history.

Furthermore, it is worth noting that trend analysis could not be done without the help of VirusTotal data. By grouping and reviewing metadata and behavioral reports from VirusTotal, researchers could uncover newly emerging tendencies in malware behavior notably those concerning persistence mechanisms. The trend analysis performed under this section enables us to understand what their evolution has been like over time in response to security improvements happening across the globe including developing bypassing techniques by tracking how each variant strategizes around new security walls or utilizes fresh security exploitation points. By doing so, not only academic field comprehend better the long-standing character of malware activity, especially regarding its resilience against potential detection mechanisms at hand but also implemented security solutions could be useful up-to-date (pointing out that) such things as patterns of growth/decay between arms manufacturers developing arms malware developers malware. Such insights would also be crucial in real-world security scenarios since they illustrate how malware is continually changing security measures and the never-ending war between cybercriminals and their adversaries within the cybersecurity sector.

Finally, this enriched dataset together with the exhaustively gathered VirusTotal data enabled a more differentiated and comprehensive examination of malware persistence ideologies; thus making all conclusions based on them firmer.

3.5. Ethical Considerations

Malware research involves handling malicious software, which poses significant ethical and legal challenges. This section outlines the steps taken to address these issues, ensuring that the research was conducted responsibly and in compliance with legal standards.

3.5.1. Compliance with Legal Frameworks

All malware samples were obtained from sources that provide them for research purposes, and all activities were conducted in compliance with the relevant legal frameworks. The use of VirusTotal, MalwareBazaar, and other repositories was done by their terms of service, ensuring that the research adhered to legal standards.

3.5.2. Responsible Handling of Malware

To mitigate the risks associated with handling malware, all samples were analyzed in isolated environments, such as virtual machines and sandboxes. This containment strategy ensured that the malware could not spread or cause harm beyond the controlled environment. Additionally, the research did not involve the dissemination or distribution of malware samples, further minimizing potential risks.

3.5.3. Privacy and Data Protection

The research did not involve any personal data, focusing solely on the technical aspects of malware. However, care was taken to ensure that all data, including metadata and behavioral reports, was stored securely and accessed only by authorized personnel. This approach safeguarded the integrity and confidentiality of the research data.

4. Analysis of Persistence Techniques in the MITRE ATT&CK Framework

The extensive information system of the MITRE ATT&CK [41] framework describes adversarial tactics and techniques that have been derived from actual observation. It is used by cybersecurity personnel in analyzing the actions of cyber threats and in the formulation of measures that can be put in place to detect and prevent these threats. Lots of efforts are paid to persistence tactics in the ATT&CK framework as those tactics would enable adversaries to hold a long-term position in the target system, meaning that the attackers would be able to continue their activities notwithstanding the reboots and other attempts from the defending side to eliminate them.

4.1. Overview of Persistence Techniques

The persistence techniques are the procedures that the attackers employ to maintain access to a given system in case there is a disruption such as rebooting or loss of terminal access. The MITRE ATT&CK framework lists the techniques and sub-techniques to offer the analysts the details of how threats persist and how to prevent them.

Some of the most notable persistence techniques in the MITRE ATT&CK framework include:

4.2. Registry Run Keys / Startup Folder (T1547.001)

Description:

Attackers alike, the Windows registry and Startup folder as a way to stay resident in a compromised computer. For instance, they may change or add subkeys to system registry keys, including HKLM Software Microsoft Windows Current Version Run, or HKCU Software Microsoft Windows Current Version Run, which runs constantly when a user signs in. Furthermore, the attackers might copy viruses, Trojans, or scripts in the startup folder, which will run at start time every time.

The method works well because the registry and the startup folder are legitimate parts of the Windows OS dedicated to the auto-launch of applications. However, when in the hands of adversaries the same mechanisms can be utilized to guarantee that commands such as executing more code is run repeatedly to maintain reinfection on the affected machine. The ability to change, as well as combining these entries and the fact that many security programs are not designed to track registry changes or modifications to the startup folder in real-time makes this technique a favorite for attackers.

Impact:

For the bots and other malware, the use of Reg Run Keys and the Startup folder to install themselves and to remain persistent is common as can be seen in all types of malware including the simpler adware to the more advanced Advanced Persistent Threats.

The main effect of this technique is that malware can be set to load with the operating system, and therefore the malicious code can go on operating each time the system restarts or a user logs in. This capability is essential for attackers who wish to have their malware persisting in the system for a very long time even after restarts or log-outs.

Furthermore, the use of this technique fully incorporates Windows system functions that make it very hard for a user to detect and remove it without assistance from a security program. The persistence provided by modifying registry keys or the startup folder enables the further execution of other heinous acts such as data theft.

4.3. DLL Side-Loading (T1574.002)

Description:

DLL side-loading is an advanced process of the adversary's attack wherein the dumb Windows application-exploitation method is utilized to launch a malicious program by controlling the way apps load DLLs. This technique is the alteration of the DLL search order or the substitution of the genuine DLL files with the forged ones. As a rule, at the start of an application, the required DLLs are loaded from certain directories in a certain sequence. This process can be exploited by an attacker by placing an attacker's DLL in the directory that an application is most likely to look for them or by renaming an attacker's DLL to a legitimate DLL name that the application is expecting to find. When it is opened, it loads the ill-intent DLL in place of the real DLL causing the attacker's code to be run using the trusted application.

Impact:

Another powerful method used by DLLs is to stay undetected in a particular system and this is known as side-loading. This technique enables the attacker to run his/her unauthorized code through a trusted system process hence not arousing the alarm at first instance. The malicious DLL works under the context of a system or application process hence making it hard to be detected by most security solutions. This technique is quite nasty due to the fact it can be used to acquire higher-level access, launch other malicious scripts, or set up a persistent presence on the suffered system in the wake of reboots and system upgrades. However, DLL side-loading can be combined with such methods as code injection as well as intensification of privileges to increase the efficiency of the attack as well as its duration. This persistence achieved through DLL side-loading is especially hard to combat, for the malicious DLL is nearly as looks identical to clean files, and, in many a case, the existence of the malicious DLL goes unnoticed by antivirus software that chiefly uses file signatures in detection.

4.4. Scheduled Task/Job (T1053.005)

Description:

Scheduled tasks or jobs are a common feature in operating systems like Windows, allowing users and administrators to automate the execution of scripts, programs, or commands at specific times or intervals. Adversaries exploit this feature by creating new scheduled tasks or modifying existing ones to ensure the regular execution of their malicious payloads. By leveraging the system's built-in task

scheduler, attackers can specify when and how often their malware should run, whether it be at startup, at a certain time of day, or after a specific system event (e.g., user login).

This technique is particularly insidious because the scheduled tasks can be configured to run with elevated privileges, enabling the malware to execute with the same permissions as a trusted system process or user. Furthermore, the scheduling process is typically transparent to the user, running in the background without any visible indication that something is amiss.

Adversaries can also set up tasks to run only when the system is idle, further reducing the likelihood of detection. The versatility and flexibility of scheduled tasks make them an attractive persistence mechanism for attackers, who can ensure their malicious code continues to execute long after the initial infection.

Impact:

The use of scheduled tasks or jobs for persistence is a powerful technique that significantly enhances the survivability of malware within a system. By setting their malicious code to run at regular intervals, attackers can ensure that their payload is executed consistently, even after system reboots or other interruptions. This regular execution makes the malware much harder to remove, as it will continue to reappear until the scheduled task itself is identified and deleted.

The ability to run with elevated privileges further increases the risk, as the malware can carry out a wide range of actions, from altering system settings to exfiltrating sensitive data, without requiring additional permissions each time it runs.

Additionally, because scheduled tasks are a standard feature of operating systems, their presence does not typically raise immediate suspicion, allowing the malware to blend in with legitimate automated tasks. This technique is especially useful in long-term campaigns, where persistence over weeks or months is necessary to achieve the attacker's objectives.

4.5. Boot or Logon Autostart Execution (T1547.001)

Description:

Boot or Logon Autostart Execution is a persistence mechanism where an attacker ensures that the system runs malicious code either at the time of booting up or during log-in. This method exploits the autostart possibilities in most operating systems: the boot configuration files, the Windows registry, logon scripts, or the startup folders. By placing the malware in these points of the system initialization, the attackers guarantee that their code is run automatically every time the system is restarted or the user signs in, without the user’s effort.

Another advantage of this technique is the flexibility of the implementation since it can be adapted to the target environment in a variety of ways to name but a few where it can be applied, operating systems, and configurations among others. Also, because most of these autostart mechanisms are intended to start essential system services and user applications, these mechanisms are allowed by the system, especially getting past the typical antiviral and firewall detectors.

Impact:

The repercussion of Boot or Logon Autostart Execution on system security is considerable since the malware is initiated the moment the system is launched, and this is most of the time before security measures such as antivirus or firewalls are activated fully. It enables the malware to set foot in the system and may even harm or neutralize protective mechanisms in anticipation of the moment when it will be noticed and dealt with. If executed at that stage of the system's life cycle it can also escalate privileges of the system, which may be used to alter the settings of the target system, add more malware, or steal data.

4.6. Hijack Execution Flow (T1574)

Description:

Hijack Execution Flow is one of the most advanced methodologies employed by adversaries where instead of allowing the legitimate process to execute its normal flow, the adversaries control the process and alter the flow to have the process execute malicious code. This technique exploits legit processes for they are trusted and hold certain privileges for execution and hence hides the execution of malware. The hijacking can be accomplished in different ways, for instance through the use of a software defect, or memory structures or through alteration of the path of execution of the target process.

Some of these techniques include the use of a technique where code is injected into the operating system process memory space, such as the DLL injection, where the attacker injects code that will be loaded by an executing process that is already running in the memory. One more type of attack is to substitute or amend a genuine executable or a library with one containing extra unauthorized functionality.

These techniques enter the execution flow of the adversary’s code, which allows the code to run within the context of a trusted process, which in the case of APIs makes it very difficult for security tools to differentiate between normal and threatening behavior. This technique has a wide application scope because it can be used at any stage of the cyber attack: getting initial access, acquiring user privileges, maintaining presence, and hiding it from the system’s owner.

Impact:

Several consequences can be derived from Hijack Execution Flow, but to be more precise, it allows the malware to run as a normal process, which makes it much more difficult to detect and analyze.

Through such redirection of the execution flow, the attacker can disrupt typical system functioning, take control over crucial operations, and possibly obtain critical information or even raise the level of their permissions. It is very useful in the way of maintaining persistence as the virus embedded itself in trusted and normal processes that should run, this goes unnoticed by antivirus software.

4.7. Service Registry Permissions Weakness (T1574.007)

Description:

Service Registry Permissions Weakness is a technique where adversaries exploit weak or improperly configured permissions on service registry keys to redirect the execution of legitimate services to run malicious code. In Windows operating systems, services are managed through registry keys that define how and when they start, the executables they run, and the privileges they use. If the permissions on these registry keys are not properly secured, an attacker can modify them to point to a malicious executable instead of the intended legitimate service.

This manipulation allows the adversary to hijack the execution of the service, causing the system to run the malicious code with the service's privileges whenever the service is started. This technique is particularly dangerous because it exploits the inherent trust in system services, which are expected to run as part of normal operating system functions. By targeting services that are automatically started by the operating system, the attacker ensures that their malware is executed reliably and often with elevated privileges, increasing the potential impact of the attack.

Impact:

The impact of exploiting Service Registry Permissions Weakness can be significant, as it allows attackers to maintain persistence within a system by leveraging existing, trusted services that the operating system routinely executes. Once the service registry key is modified, the malware can be executed automatically each time the service is triggered, whether at system startup, upon user login, or when the service is manually started.

This method is particularly effective because it does not require the creation of new services or processes, which might raise suspicion; instead, it hijacks an already existing, legitimate service. This approach makes the malicious activity harder to detect and increases the likelihood that the malware will go unnoticed for extended periods. Furthermore, because services often run with elevated privileges, the malware can perform a wide range of actions, from modifying critical system settings to accessing sensitive data.

The use of this technique also complicates the process of removing the malware, as simply deleting the malicious executable may not be sufficient if the service registry key is not also restored to its correct configuration. The persistence gained through this method can be particularly resilient, allowing the malware to survive system reboots and attempts to disable or remove it.

Overall, exploiting service registry permissions weaknesses is a powerful strategy for adversaries seeking to maintain a covert and durable presence within a compromised system.

5. Case Studies / Analysis

5.1. Analysis of Database Structure

The structure of the database comprises three main tables as follows:

Figure 4 - Database Structure for the Malware Techniques Analysis

Table all_hashes

id: Integer number used as a primary key

hash: Variable character field used to store hash values representing malware samples

Table hash_info

id: Integer number used as a primary key

hash: Variable character field linking to hashes in the all_hashes table

signature: Variable character field containing signature or name associated with the malware hash

Table malware_info

id: Integer number used as a primary key

mal_hash: Variable character field linking to hashes in the all_hashes table

detections: Longtext field, where one may expect to find some information about detections associated with the malware

mitre_att_techniques: Longtext field that is thought to contain the list of MITRE ATT&CK techniques related to this malware.

This structure supports the storage and retrieval of data on malware. All_hashes acts as a starting point because hash_info and malware_info provide additional details that can be queried based on their values in hashes.

5.2. CSV Files Analysis

CSV files which you mentioned earlier are purposed for conducting statistically meaningful analyses as well as visualizing malware persistence techniques. Here is how one can use this data:

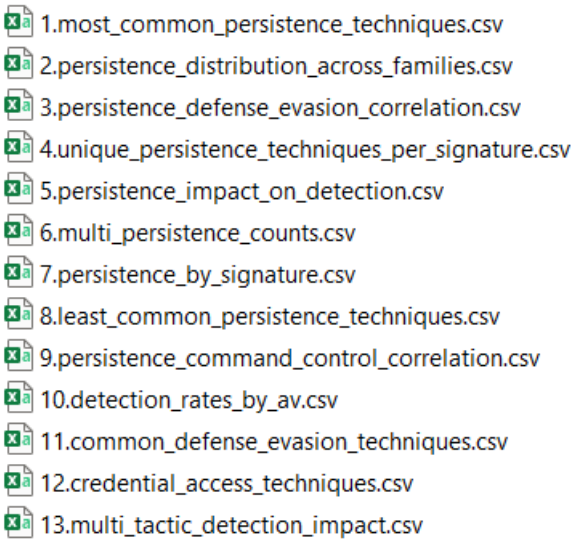

Figure 5 - CSV Files for categorization and correlation Analysis

Most Common Persistence Techniques

A file listing various persistence techniques and their frequencies. This file can help one to realize the most common forms of persistence among malware, thus being able to predict their behaviors for effective countermeasures.

Persistence Techniques by Signature

This document maps out a specific signature for each instance of malware, along with the techniques used to make it persistent. For example, military-grade algorithms or hats control communications lines during attacks through radio signals since they remain undetectable at first glance but have high power levels, making them hard targets.

Trend and Correlation Analysis

These include files like *persistence_defense_evasion_correlation.csv* and, *persistence_command_control_correlation.csv* which aim at correlating these types of operations. It's true that through understanding these patterns' law enforcement agencies and national security organizations continue tracking suspects involved in terrorism-supporting activities online/offline

across nations worldwide since the internet provides such people with anonymity therefore enabling secure communication channels between them.

Statistical and Visualization Outputs

These files are typically in diverse CSV formats depicting various counts, distributions, and correlations used to feed visualization scripts that create graphs and charts that reveal significant patterns within data.

For example, a file named "*persistence_impact_on_detection.csv*" can be used to plot how various types of persistence affect the detection rates of antivirus programs.

Python Scripts

Those Python scripts you have were created for specific purposes concerning extraction, processing, and analyzing data; here is an outline of what would possibly entail.

5.3. Data Collection

Automating the process of downloading malware samples along with their corresponding hashes from different malware sources are some of the major goals these python scripts aim to achieve, which include *collect_hashes_mal_bazaar.py*, *collect_hashes_virus_share.py*, and *collect_hashes_zoo.py*.

Collect_hashes_mal_bazaar.py python script is designed to interact with the MalwareBazaar API to retrieve malware-related hashes based on different tags, store these hashes in a file, and ensure that there are no duplicate entries

File "*collect_hashes_virus_share.py*" is designed to download hash files from the VirusShare website and save them to a specific directory on your local machine. The script automates the process of downloading different hash files from VirusShare. These files possibly have the MD5 hashes of samples of malicious programs that can be helpful in the security field, for malware analysis or other cybersecurity activities. The script makes sure that all given files are downloaded and saved in an orderly manner, taking into consideration cases when some folders have disappeared or downloads have failed.

The file "*collect_hashes_zoo.py*" is the script that automates the process of downloading a large collection of malware samples from the "TheZoo" repository, extracting SHA-256 hashes from these samples, and storing them in a single text file. This is useful for researchers, analysts, and anyone working with malware data who needs a consolidated list of malware hashes for further analysis, matching, or threat intelligence purposes.

File "*collect_malware_info*" is a script provided to process and clean malware-related data stored in a database, focusing on extracting specific information such as persistence techniques and antivirus detections. It connects to a database using credentials from a configuration file, loads relevant tables into data frames, processes JSON data related to MITRE ATT&CK techniques and antivirus detections, and merges this information with hash signatures. Finally, the cleaned and enriched data is saved into

CSV files for further analysis or reporting. The script automates the extraction of critical insights from raw malware data, streamlining the analysis process for cybersecurity professionals.

This ensures that unique hashes remain by merging datasets using combine_hashes.py or combine_virusshare_hashes.py to combine these two files and remove duplicates present in either file Data Processing. To get a specific malware signature from some outside sources like VirusTotal then add this information into a database is carried out by *get_hash_signature.py* and *insert_hashes_to_db.py* scripts. There is also a script named *reduce_hashes_based_on_date.py* which helps filter out specific criteria from the dataset such as focusing more on recent samples among others Data Analysis and Visualization.

These scripts include *insight_generation.py* and plot_creation.py which perform statistical analysis on the processed data and generate visualizations out of it. The aim is to create documents that use clean data ready for reporting or other kinds of analysis Next Steps. To explain in more details for each script, a short description is provided:

insert_hashes_to_db.py: This script processes a list of hashes from a file and inserts them into a MySQL database. It reads the hashes, connects to the database, and inserts each hash into the all_hashes table, ensuring that duplicate entries are handled appropriately.

reduce_hashes_based_on_date.py: This script filters malware hashes based on the date they were first seen, as reported by the MalwareBazaar API. It reads a list of hashes, queries the API to retrieve the date for each hash, and saves only those hashes that meet the specified date criteria to a new file.

get_hash_signature.py: This script retrieves signatures for malware hashes by querying the MalwareBazaar API. It reads hashes from a file, queries the API for each hash to obtain its signature, and stores the hash-signature pairs in a MySQL database, aiding in the classification and analysis of malware samples.

rescan_hashes_VT.py: This script interfaces with VirusTotal's API to rescan and retrieve detailed reports for a range of malware hashes stored in a MySQL database. It extracts information such as antivirus detections and MITRE ATT&CK techniques, and saves this data back into the database, enriching the malware data with up-to-date threat intelligence.

insight_generation.py: This script processes malware data stored in a CSV file to generate various insights related to persistence techniques, defense evasion, and other tactics. It performs analysis on these techniques, calculates their frequency and impact on detection rates, and outputs these insights into new CSV files, facilitating further analysis or reporting.

plot_creation.py: This script generates visualizations from the insights data produced by the insight_generation.py script. It creates various plots such as bar charts and heatmaps that illustrate the prevalence and correlation of different malware techniques, as well as the impact of these techniques on detection rates. The plots are saved as image files for easy reference and presentation.

reform_zoo_hashes.py: This script is used to extract and clean hash values from text files within the "TheZoo" repository. It processes these files to isolate unique hash values, which it then saves in a sorted format to a new text file, ensuring a clean and organized dataset of malware hashes.

6. Results

6.1. Most Common Persistence Techniques

The following chart illustrates the most common persistence techniques employed by malware, with the techniques listed on the y-axis and their respective occurrence counts on the x-axis. Notably, "Hijack Execution Flow" and "DLL Side-Loading" emerge as the most prevalent techniques, each exceeding 5000 occurrences.

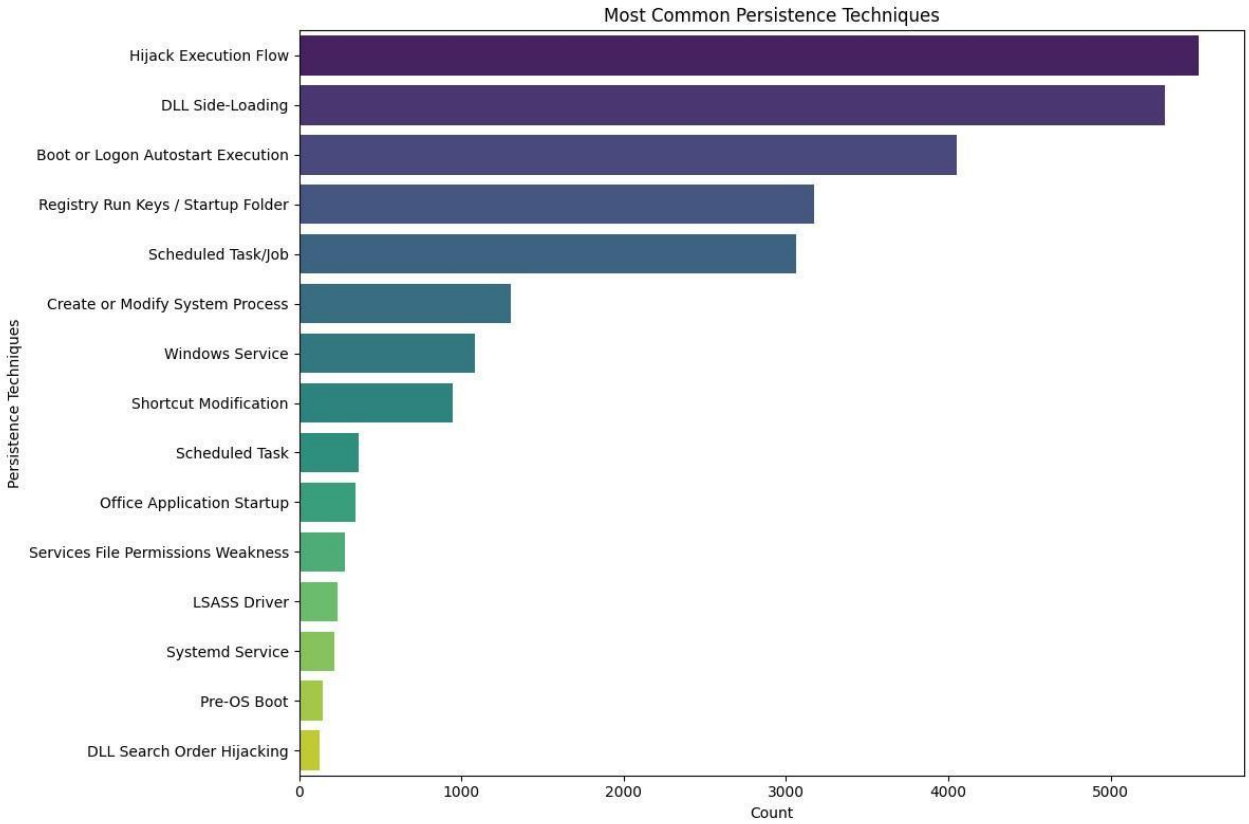

Figure 6 - Most Common Persistence Techniques

These methods involve altering the normal execution flow or loading malicious DLLs into processes to maintain persistence. "Boot or Logon Autostart Execution" and "Registry Run Keys / Startup Folder" also feature prominently, suggesting that malware often targets system startup processes or modifies registry entries to ensure execution during system boot or user logon. Additionally, "Scheduled Task/Job" is a commonly used method that enables malware to persist by scheduling tasks to run automatically at designated times or intervals. Although less frequent, techniques such as "DLL Search Order Hijacking," "Pre-OS Boot," and "Systemd Service" still warrant attention.

The plot underscores that malware primarily relies on hijacking execution flows and manipulating DLL loading processes to achieve persistence, likely due to their effectiveness and the challenges in

detection. Understanding these techniques is critical for crafting effective detection and mitigation strategies to thwart malware from establishing a long-term presence on compromised systems. The diversity of techniques also highlights the importance of comprehensive security measures that address multiple persistence vectors.

6.2. Top Persistence Techniques Across Selected Signatures

The heatmap in the following figure visualizes the distribution of top persistence techniques across various malware signatures, with the y-axis listing different malware types such as AgentTesla and QuasarRAT, and the x-axis representing distinct persistence techniques. The color intensity indicates the frequency of each technique within each malware signature, with darker shades representing higher counts.

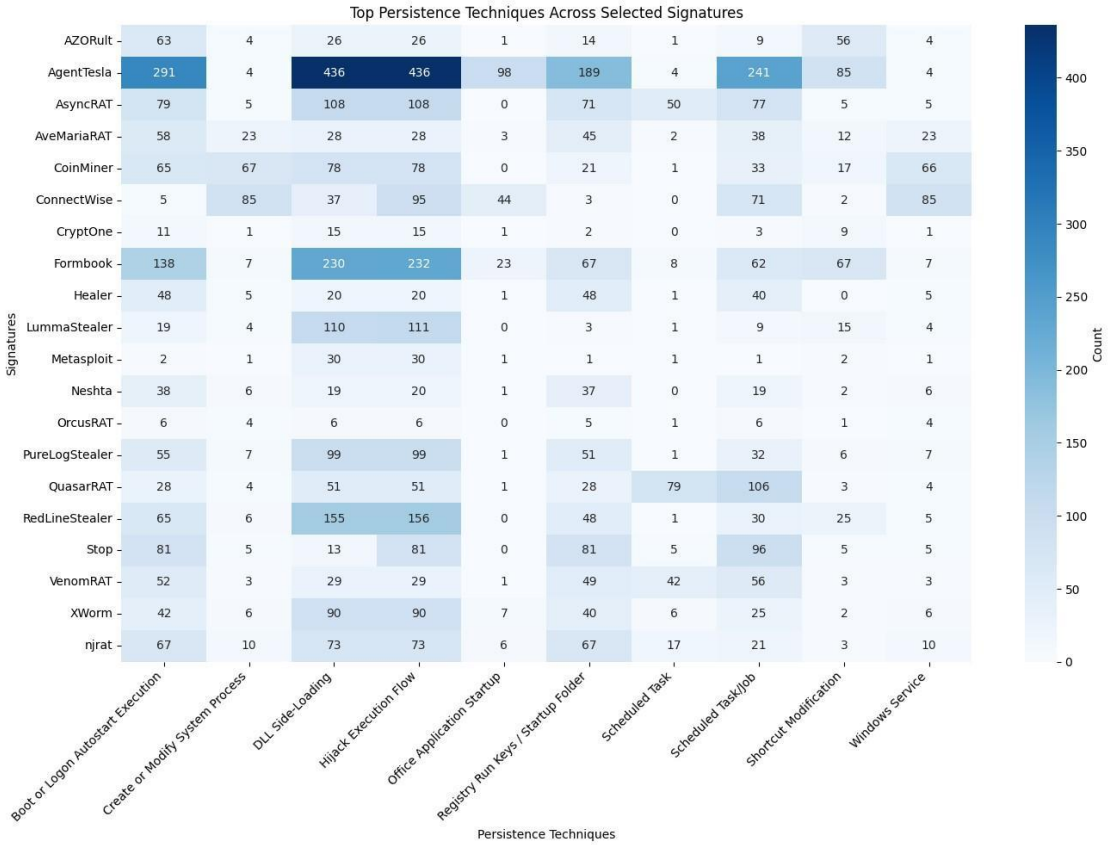

Figure 7 - Top Persistence Techniques Across Selected Signatures

AgentTesla emerges as one of the most versatile malware families, extensively using techniques like "DLL Side-Loading," "Hijack Execution Flow," "Registry Run Keys / Startup Folder," and "Scheduled Task." These methods are central to its persistence strategy, with some techniques appearing over 400 times.

Similarly, RedLineStealer and Formbook show a strong reliance on "Hijack Execution Flow" and "DLL Side-Loading," suggesting a shared approach to achieving persistence. QuasarRAT, on the other hand, focuses heavily on "Registry Run Keys / Startup Folder" and "Scheduled Task," indicating a preference for methods related to system startup and task scheduling. Malware such as AZORult and XWorm make

significant use of the "Windows Service" technique, which ensures the malware runs continuously by configuring system services.

In contrast, less sophisticated or less widespread families like OrcusRAT and Metasploit show minimal use of multiple persistence techniques, possibly reflecting a more targeted or specialized application.

In conclusion, the heatmap highlights that certain malware families, particularly AgentTesla, RedLineStealer, and Formbook, employ a diverse array of persistence techniques, making them more resilient and difficult to eradicate. This flexibility in using various techniques likely contributes to their success in remaining undetected in infected systems. The consistent application of methods like "Hijack Execution Flow" and "DLL Side-Loading" across multiple malware families underscores their effectiveness. Additionally, the dependence of specific families on techniques such as "Registry Run Keys / Startup Folder" suggests that focusing on these areas in security strategies could be key to mitigating the impact of certain malware types. Overall, the plot emphasizes the need for comprehensive security measures capable of detecting and countering a broad spectrum of persistence techniques across different malware signatures.

6.3. Correlation Between Selected Persistence and Defense Evasion Techniques

The following figure shows the correlation between chosen persistence methods intact by malware from this website and some defense evasion methods giving them a tough time. On the y-axis, we have listed several kinds of persistence techniques, while on the x-axis there are different categories in which defense evasion tactics fall. Darker patches on this heat map signify a higher frequency of each combination of techniques.

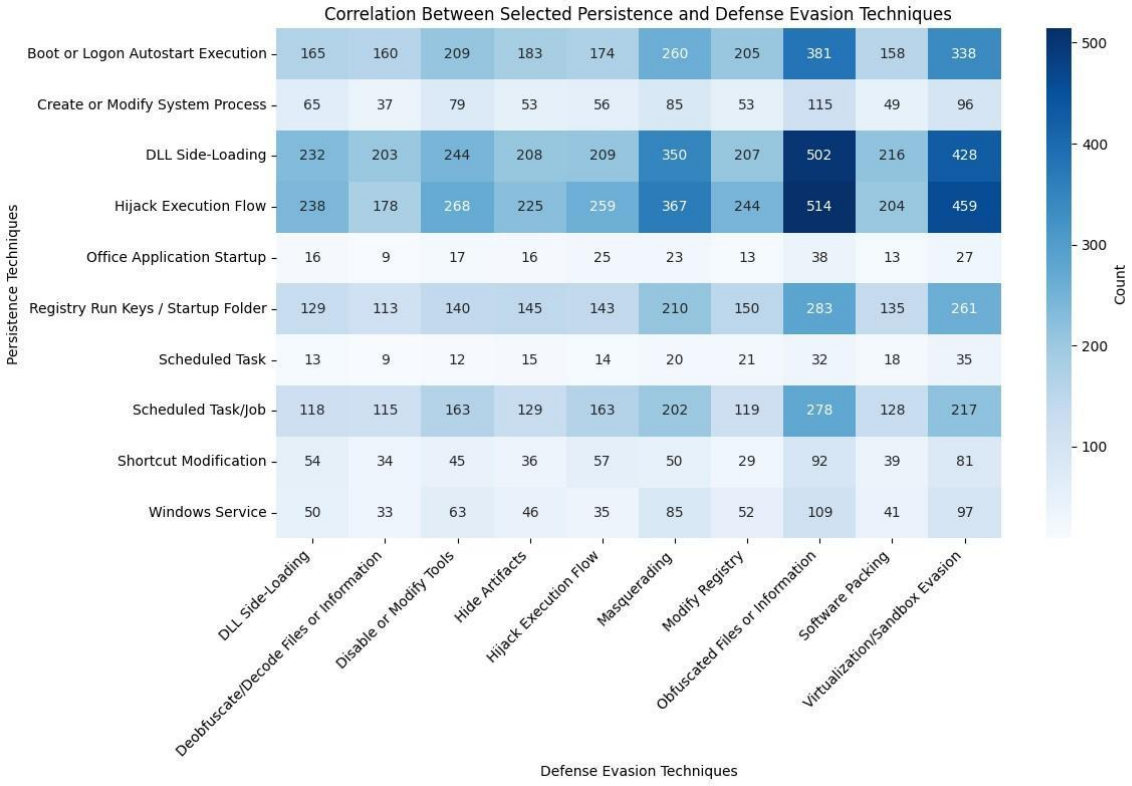

Figure 8 - Correlation Between Selected Persistence and Defense Evasion Techniques

If we look at the relationships between DLL Side-Loading and Hijack Execution Flow there is a good match between them and several defense evasion techniques such as hiding artifacts, obfuscated files/information as well as virtualization/sandbox evasion. Therefore, it can be argued that whenever malware uses any of these methods to stay in a computer system without being removed or analyzed, it employs other advanced mechanisms for preventing detection and analysis processes from finding it.

The other pairings include Boot or Logon Autostart Execution with Obfuscated Files or Information, and Registry Run Keys / Startup Folder with Virtualization/Sandbox Evasion. This means that whenever malware uses methods that will make it start up when the system boots or during user logon, it will also employ advanced evasion techniques to ensure that it cannot be detected at these points.

Scheduled Task/Job significantly correlates with Hijack Execution Flow and Modify Registry in this way, as these may be used together to maintain system access points by combining scheduled tasks for persistence with registry modifications and execution hijacking by the malware.

Windows Service has a mild relationship to several obfuscation methods, particularly to “Obfuscated Files or Information” and “Virtualization/Sandbox Evasion,” pointing out the commonality between hiding measures and the virtual machine escape technology within its context.

Observing the diagram, seems that Malware usually makes sure it links persistent methods with robust defense evasion plans in order not only to work but also to avoid detection. This is a clear demonstration of high-level obfuscation mechanisms that make popular evasion scenarios such as DLL side-loading or process hijacking notable examples of persistence strategies integrated with modern antivirus immunity options like full-system scans for rootkits; besides, when we examine how Obfuscated Files & Information and Virtualization/Sandbox Evasion appear next to Boot or Logon Startup Modules you suddenly realize that security products must deal with both issues simultaneously if we are going fight back against it. In general terms, this illustration stresses that elaborate malware programs deal with two-dimensional issues concerning persistence and avoidance, thereby making cybersecurity as a whole should be comprehensive enough to include these ideas.

6.4. Top Unique Persistence Techniques by Top Signatures

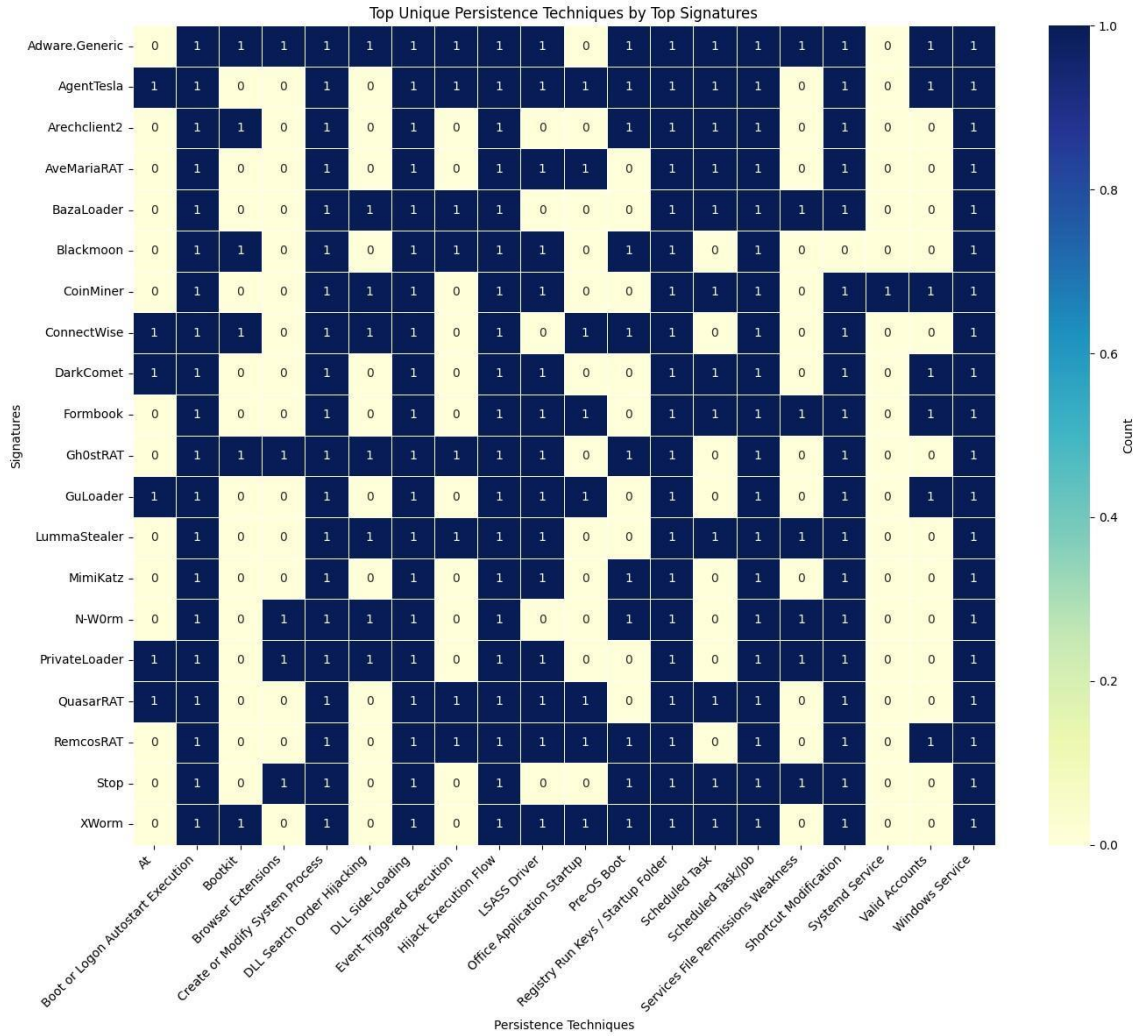

Figure 9 - Top Unique Persistence Techniques by Top Signatures

This heatmap illustrates the unique persistence techniques employed by various top malware signatures. The y-axis lists different malware signatures, while the x-axis represents specific persistence techniques. The color scale indicates the presence (1) or absence (0) of each persistence technique within a given malware signature, with darker shades corresponding to the presence of the technique.

The heatmap reveals that certain persistence techniques are widely shared across multiple malware signatures. Techniques such as "Registry Run Keys / Startup Folder," "Scheduled Task/Job," "Hijack Execution Flow," and "DLL Side-Loading" are commonly utilized, appearing in nearly all the listed malware families.

AgentTesla, **QuasarRAT**, and **RemcosRAT** display a comprehensive use of various persistence techniques, suggesting that these malware families are highly versatile in maintaining persistence within a system. Their strategy appears to involve multiple redundant methods to ensure they remain operational despite potential detection efforts.

Conversely, some malware families, such as **Adware, Generic,** and **BazaLoader**, show a more limited range of persistence techniques. This could imply a more focused or less sophisticated approach to persistence, possibly making them easier to detect or remove.

The uniformity of technique usage across many malware families suggests a standardized approach to persistence, where specific methods have proven highly effective and are thus adopted widely across different malware types.

6.5. Top Persistence Techniques Impact on Detection Rates

This graph (Plotting Graph) shows how different persistence techniques affect detection rates. The x-axis shows various techniques, while the y-axis displays average detection rates in percentage form. Mostly, when a certain persistence technique is commonly recognized by security tools, it indicates more detection instances as represented by higher bars.

Notably, in the case of the Path Interception through Search Order Hijacking technique it has the highest average detection rate, implying that most security mechanisms are aware of it due to its widespread knowledge in addition to the reliable means used in detecting it.

Taking the example of Active Setup, AppInit DLLs, and Winlogon HelperDLLs, all these methods have high rates of detection showing that they are mostly listed as high-risk activities of system starting up and manipulating of dynamic linked libraries by security solutions. Otherwise, refer to Account Manipulation and System Firmware manipulation techniques which involve more sophisticated modifications on user's accounts or motherboards in general have medium rates of detection due to ongoing improvements in the field of IT security tools that identify them.

However, methods like creating an Account or Changing Folder Associations were slightly lower than most others for reasons that could be due to their complex nature or because they are not as well covered by other programs available today. This figure suggests that while some of these tactics can be efficiently detected; others might not always be caught. The conclusion here is that we still need better detection capabilities covering more techniques, thereby leading to sturdy defense measures against routine and rare persistence strategies.

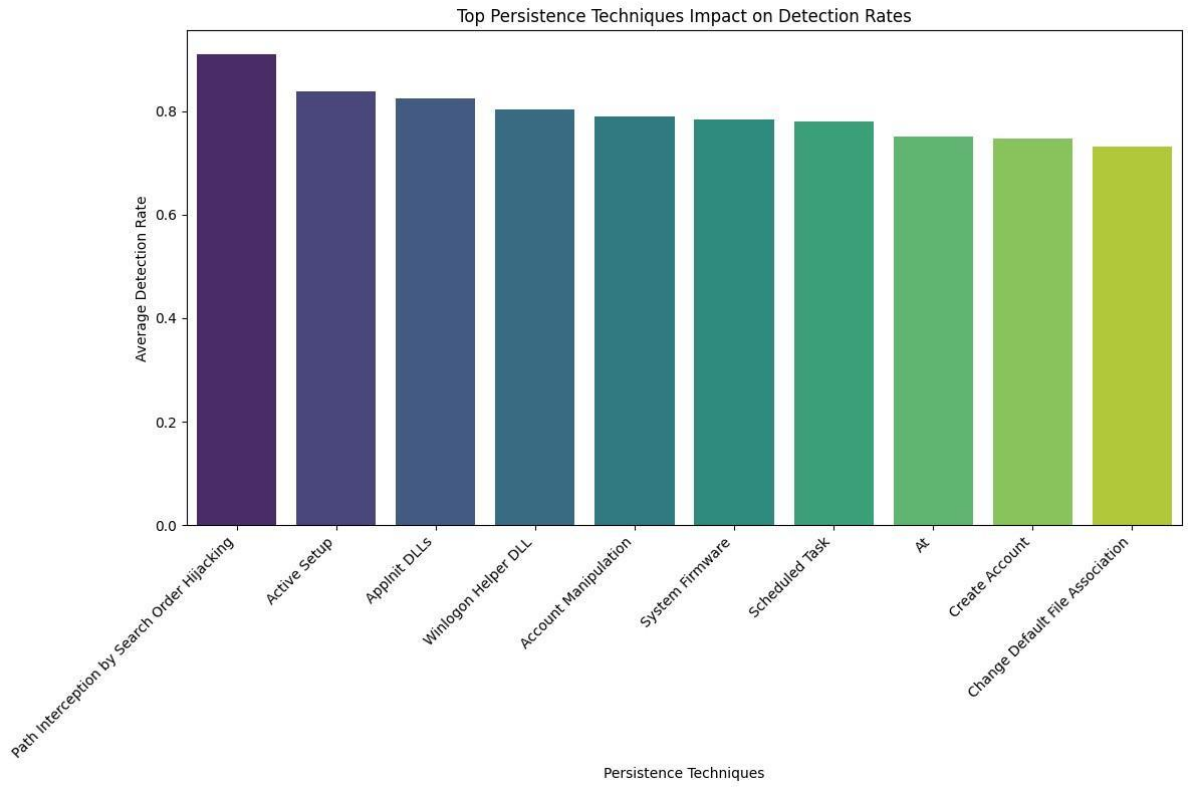

Figure 10 - Top persistence impact on detection

6.6. Frequency of Multiple Persistence Techniques in a Single Malware Sample

The following chart displays the frequency of the number of persistence techniques used in individual malware samples. The x-axis represents the number of persistence techniques employed by a single malware sample, while the y-axis indicates the count of malware samples utilizing that number of techniques.

The majority of malware samples, totaling 9,331, do not use any persistence techniques, which could imply that these samples rely on other methods for their effectiveness or do not require persistence due to their operational nature. A significant number of malware samples, 3,806, utilize exactly two persistence techniques, followed by smaller but notable counts for those employing three (1,311) and four (1,471) techniques.

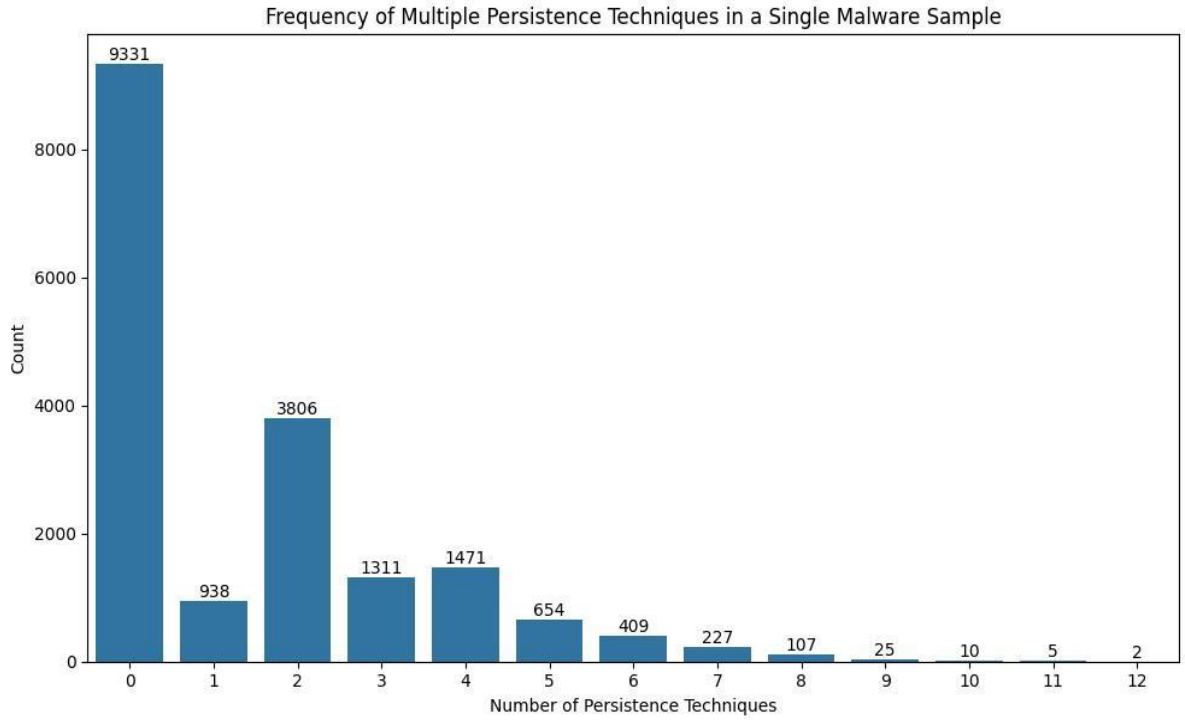

Figure 11 - Frequency of Multiple Persistence Techniques in a Single Malware Sample

This suggests that while some malware may rely on a single method of persistence, there is a substantial portion that implements multiple techniques to ensure resilience and longevity within the infected system. The usage of five or more persistence techniques in a single sample is considerably less common, with the count gradually decreasing as the number of techniques increases. This trend indicates that while malware can employ multiple persistence methods, such complexity is relatively rare, likely due to the increased sophistication and effort required to implement and manage numerous techniques effectively.

The chart highlights that most malware samples either do not use persistence techniques or rely on a limited number of them, with a sharp drop-off in frequency as the number of employed techniques increases. This suggests that while some malware may be highly sophisticated, with multiple methods to maintain persistence, the majority tends to use simpler approaches, possibly balancing effectiveness with the complexity of implementation. Security strategies should, therefore, focus on detecting and mitigating the more common single or dual-technique approaches, while also being mindful of the potential for more complex, multi-technique threats.

6.7. Comparison of Persistence Techniques Across Selected Signatures

A comparison of various malware signatures by the use of different persistence methods is presented in this heatmap. The vertical axis presents several different known malware signatures ranging from top to bottom, while the horizontal axis depicts each specific technique used to maintain itself in the system. In each cell, shade represents how often a given technique was applied by the particular malware signature, with dark colors indicating greater use.

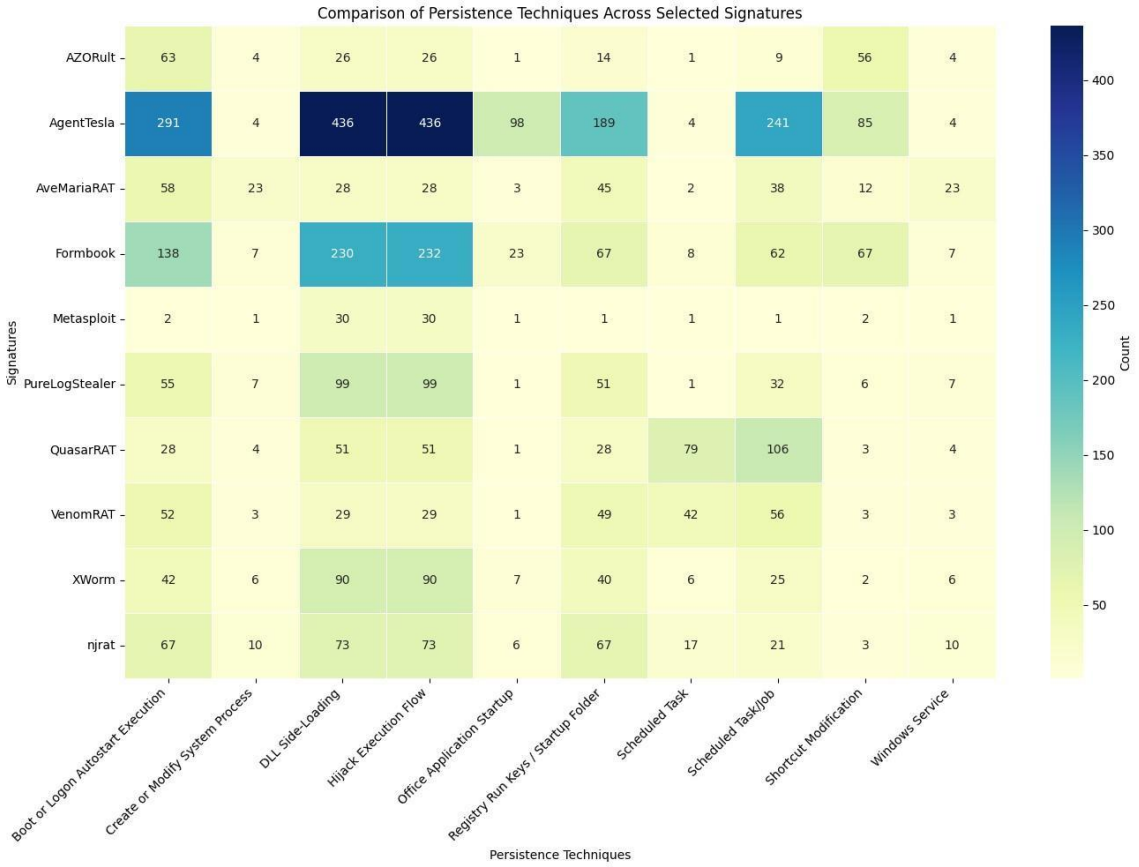

Figure 12 - Comparison of Persistence Techniques Across Selected Signatures

AgentTesla heavily depends on “DLL Side-Loading” and “Hijack Execution Flow,” which have the highest counts among the signatures shown on them. This also means that it largely uses “Registry Run Keys / Startup Folder” as well as “Windows Service” which enables it to be persistent in many ways.

Formbook, on the other hand, exhibits considerable affinity for the ‘Hijack Execution Flow’ and ‘DLL Side-Loading’ modes, suggesting that they may be sharing the same persistence mechanisms with AgentTesla. Additionally, this malware relies mostly on “Registry Run Keys / Startup Folder” and “Scheduled Task/Job” which makes it further entrenched.

Other malware like PureLogStealer, QuasarRAT, and XWorm do use “Hijack Execution Flow” heavily but apply it differently from AgentTesla in terms of less use of other techniques such as ‘Registry Run Keys / Startup Folder’ alongside ‘Scheduled Task/Job’. In particular, for PureLogStealer there are many alternatives including Creative Installer and Keylog-FTW concerning its distribution.

AZORult, VenomRAT, and njrat apply a wide range less intensely of various persistence techniques such as “Boot or Logon Autostart Execution” and “Windows Service”. Thus, these malware types are somewhat persistent across the entire spectrum of methods used by it during the targeted system’s life these days.

From the results, we can conclude that particular malware families, such as AgentTesla, and Formbook, strictly depend on several unique forms of persistence methods for example “DLL Side-Loading” and “Hijack Execution Flow” (these techniques are their key pillars for staying hidden and running). They

indicate a rigorous layered approach along with other mechanisms like ‘Registry Run Keys/Startup Folder’. On the other hand, AZORult or VenomRAT embrace a more varied but less efficient persistence strategy. This implies that differentiating between various Malware Families in terms of their flashing skills is crucial for successful detection and mitigation appropriate to each of such threats hence typed.

6.8. The Least Common Persistence Techniques

In the bar chart, the least common persistence techniques used by malware are depicted; the x-axis shows the occurrences of each technique with the count, while the y-axis shows names of specific techniques for persistence. All listed techniques in the chart are characterized by very low occurrences of malicious software samples shown in the chart. Each of the Authentication Package, Network Provider DLL, and Screensaver are rarely used in analyzed malware samples, once. They might be less popular than other methods because they are specific or there are more effective substitutes.

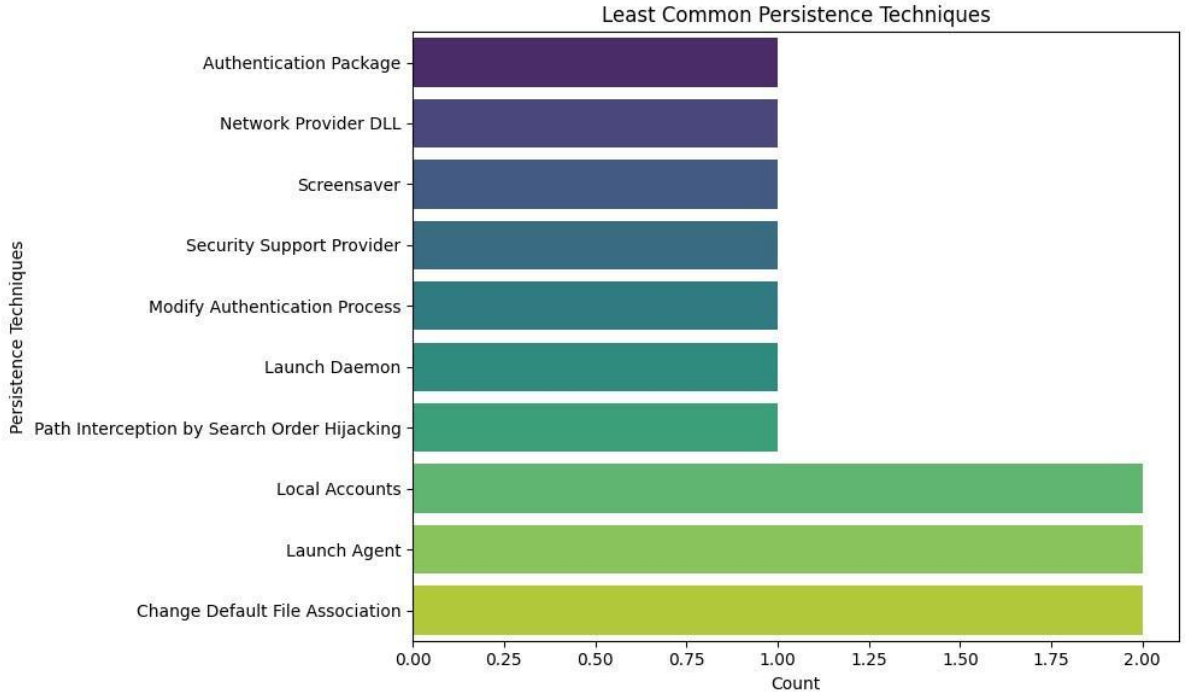

Figure 13 - Least Common Persistence Techniques

Security Support Provider and Modify Authentication Process also have low instances showing that they are not the most preferred when it comes to how persistence is maintained by malware for example by making the system less secure, but they are unfriendly particularly to most coders as they may contain complex codes.

On the other hand, Launch Daemon, Path Interception by Search Order Hijacking, and Local Accounts are slightly more frequent though among the least popular ones; hence might be applied under special circumstances or by certain types of malware.

Change Default File Association, Launch Agent and Local Accounts are the most common among the least common techniques, each appearing twice. This small rise means that such methods could sometimes be prudent for given varieties under various conditions.

Thus, the table shows that some modes of persistence are rarely used in data invasion programs; the implication is that concerning other known modes, these may be too specialized or may simply not work well in general. Authentication Package and Network Provider DLL indicate rare cases where they are adopted as primary methods for maintaining access rights on a computer, probably because they require expertise or can be effective only in certain environments. However, Change Default File Association and Launch Agent are the most recurrent way among uncommon ones, implying that albeit not widely spread, on a few occasions they can be meaningful in certain contexts. Security professionals need to be conscious of these rare strategies since their infrequency might result in lower concentration on defense mechanisms, thereby allowing a greater number of sophisticated or targeted malware to enter without being detected.

6.9. Relationship Between Selected Persistence Techniques and Command & Control

The heatmap was used to guide the exploration of the connection between chosen persistence techniques and malware’s use of command and control (C2) techniques. On this heatmap, different C2 techniques are illustrated horizontally, while a variety of persistence methods are represented on the vertical axis of the chart. This means that the darker the color of a cell is, the more frequent the occurrence of a certain pair of persistence and C2 techniques.

Hijack Execution Flow, DLL Side-Loading, and Boot or Logon Autostart Execution are the most commonly encountered persistence techniques which are also associated with several C2 techniques. In other words, this implies that these methods remain important for facilitating communication between malware and its command center even as they guarantee the longevity of such infectious agents on compromised devices through processes like hiding within the Windows' root files as well as reinstallation in case their registry entries were removed.

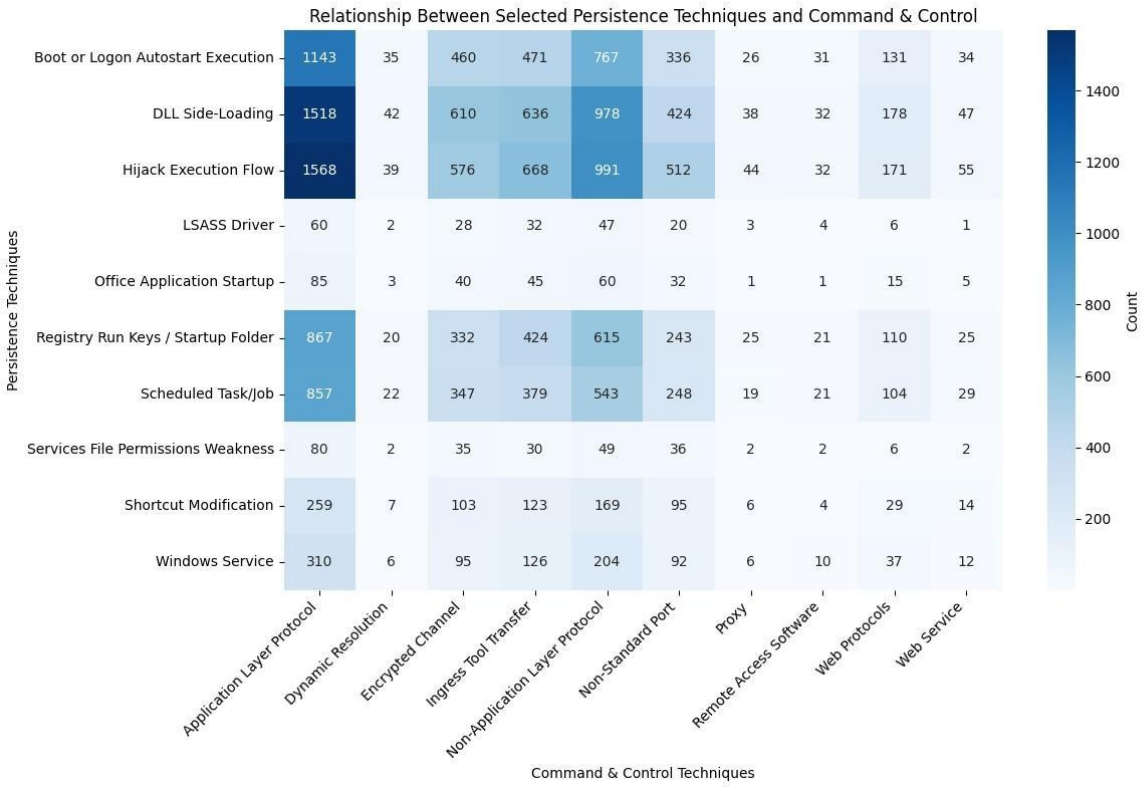

Figure 14 - Relationship Between Selected Persistence Techniques and Command & Control

Registry Run Keys / Startup Folder and Scheduled Task/Job also have strong links with some types of C2 methodologies, especially those involving 'Non-Application Layer Protocol' and 'Encrypted Channel'. Consequently, it is speculated herein that such methods are often utilized by malware requesting stable communication with remote servers to take instructions or steal information from them.

However, this is not the case when it comes to some less often observed persistence techniques. The correlation between Office Application Startup, Services File Permissions Weaknesses, and C2 shows that these might be used in certain specific or rare malware entities. For example, a moderate level of relationship between Windows Service and Shortcut Modification can be seen across various C2 methods, particularly 'Non-Application Layer Protocol' and 'Encrypted Channel'. This indicates that although they are not dominant methods of persistence, they contribute to the continuous C2 communication in certain malware samples.

In summary, the heatmap vividly demonstrates that some specific persistence mechanisms, namely Hijack Execution Flow, DLL Side-Loading, and Boot or Logon Autostart Execution, tend to go hand in hand with numerous command and control techniques. Thus, such an approach secures communication between the malware and its command center, at the same time keeping an unwanted presence on the system of an affected user. In this context, therefore, it is imperative for cybersecurity strategies to address both C2 and persistence as joint forces, since when combined they make malware more resilient and stealthy. The findings concerning different correlations further show that there are diverse strategies used by various families of malware, and thus comprehensive detection and mitigation approaches should be developed that take into account persistence and command and control relationships.

6.10. Top 20 Detection Rates Across Different Antivirus Vendors

The following chart shows the detection rates of the top 20 antivirus vendors; on the x-axis, you have the detection rate (ranging from 1) while the y-axis lists various antivirus solutions. The malware vendors are ranked from highest to lowest in terms of ability to detect and flag malicious samples.

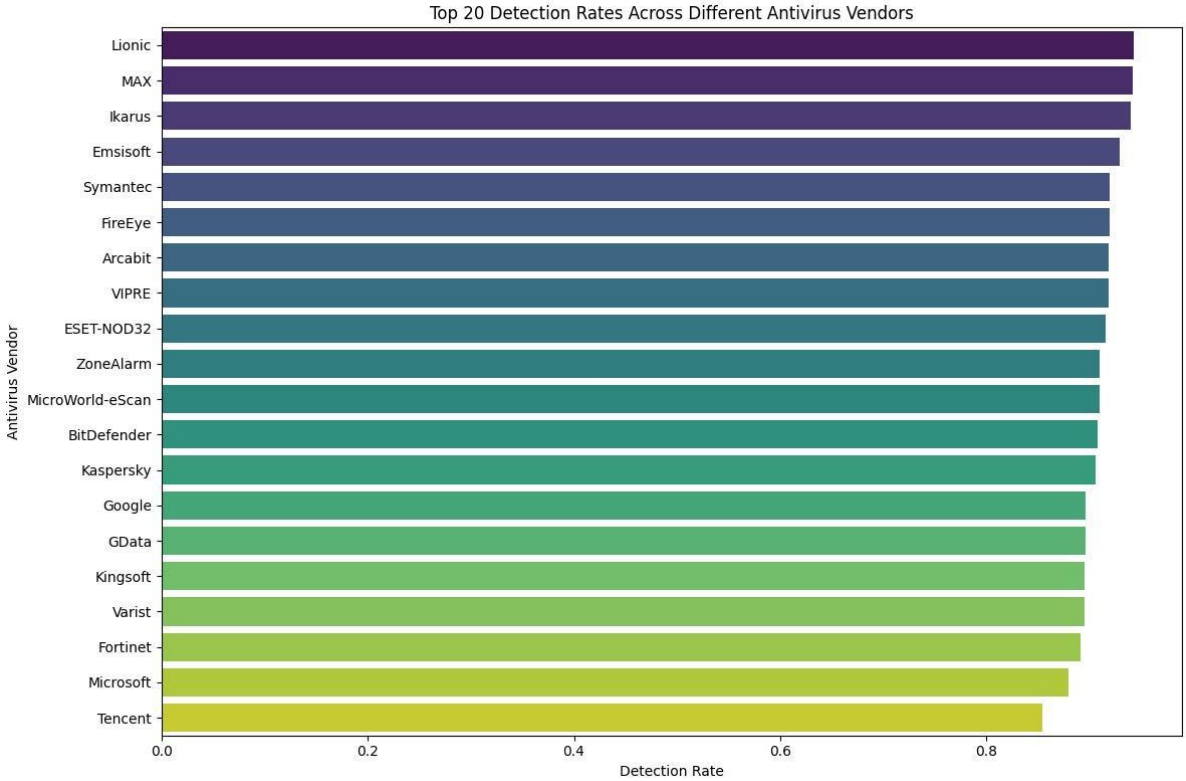

Figure 15 - Top 20 Detection Rates Across Different Antivirus Vendors

The top three vendors in terms of detection rates are Lionic, MAX, and Ikarus, showing that they are the most effective at detecting malicious programs and preventing their execution. These three always have higher rates for identifying programs that are harmful than the rest of the 20.

In addition to the above, Emsisoft, Symantec, and FireEye also exhibit notable detection capacities, placing them among the best-performing companies when it comes to this task.

Although appearing towards the bottom half of the list of 20; Microsoft still maintains reasonable rates, which shows that the lower quarter still competes fairly well against the rest of the market.

Since these optimal virus detectors are rather closely clustered together, it means that their performance is equivalent except under specialized circumstances or environments.

We can conclude that Lionic, MAX, and Ikarus are outstanding performers in dealing with malware as depicted in the chart on the vendors. A slight variation across detection rates may not necessarily mean much, since most top-end antivirus programs offer very high levels of security to your system. The

competitive nature of the antivirus flagger business is more pronounced, where vendors must always find ways to improve their system's effectiveness if they are going to stay in competition at all times. This means that users and organizations looking for these companies would rather look at other factors like specific feature sets rather than just how well can it detect the virus, this is because these will come with different performance costs.

6.11. Top 10 Common Defense Evasion Techniques

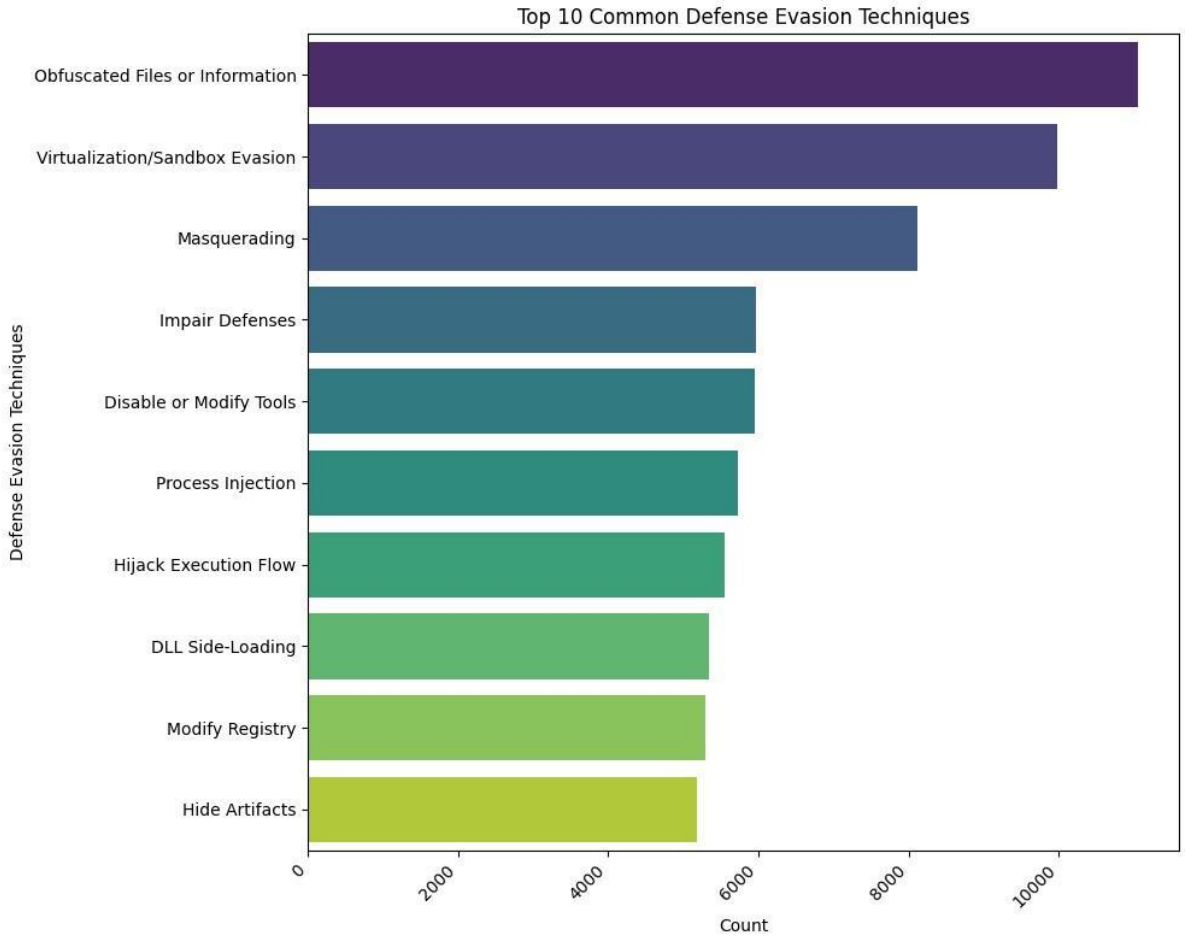

Figure 16 - Top 10 Common Defense Evasion Techniques

The x-axis of this bar graph shows the top ten most used techniques by malware for avoiding detection, while the y-axis reveals how many times each technique was seen in reports. On this chart, we see that some of the methods most commonly employed by malware include such actions as

- hiding from antivirus software, firewalls, or other protective software applications
- changing its code so that it would not be recognized as harmful or suspicious software by these same programs
- blocking any responses made against it while still operating normally, and otherwise trying anything conceivable within their power not to let anyone know what is going wrong inside of them.

6.12. Top 10 Prevalence of Credential Access Techniques

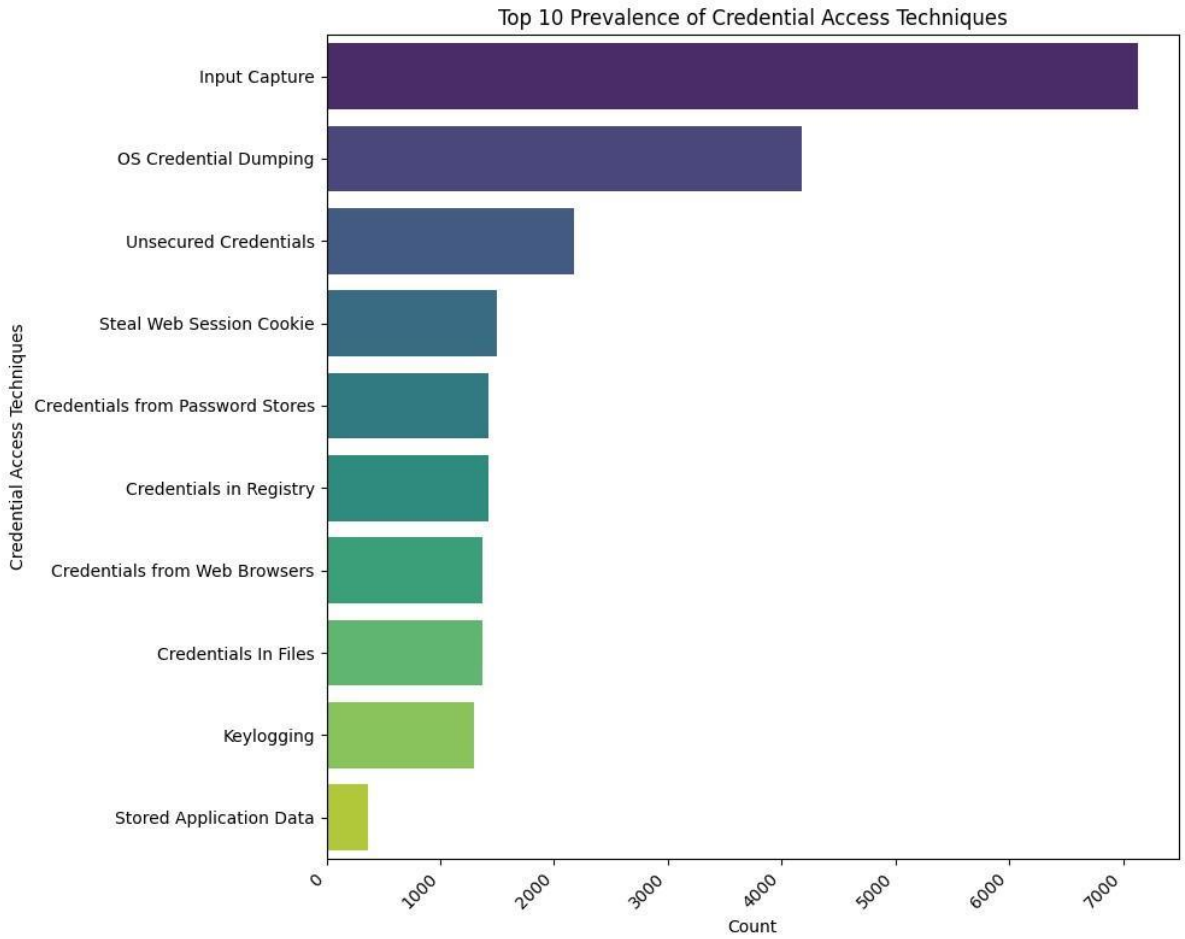

Figure 17 - Top 10 Prevalence of Credential Access Techniques

The chart above highlights the top 10 credential access techniques used by malware to obtain sensitive information. The x-axis represents the count of occurrences for each technique, while the y-axis lists the specific credential access techniques. The chart demonstrates which methods are most commonly employed by malware to gain access to user credentials.

- **Input Capture** is the most prevalent technique, with a significantly higher count than the others. This method involves intercepting user inputs, such as keystrokes, to gather sensitive information like passwords or credit card numbers, making it a powerful tool for attackers.
- **OS Credential Dumping** is the second most common technique, indicating that many malware variants attempt to extract credentials directly from the operating system's memory or security databases, such as the SAM (Security Accounts Manager) on Windows systems.
- **Unsecured Credentials** and **Steal Web Session cookies** are also frequently used methods, with the former exploiting poorly protected or stored credentials and the latter capturing session cookies to hijack web sessions and impersonate users.
- Techniques such as **Credentials from Password Stores** and **Credentials in Registry** are employed to extract stored credentials, either from password managers or system registries,

respectively. These methods underscore the risks associated with storing sensitive information in easily accessible locations.

- **Credentials from Web Browsers** and **Credentials In Files** are similarly common, targeting credentials stored in web browsers or plain text files on the system.
- **Keylogging**, a method that captures every keystroke made by the user, remains a notable but less common technique, likely due to its intrusive nature and the risk of detection.
- **Stored Application Data** is the least prevalent technique among the top 10, indicating that while some malware may target data stored within applications, this approach is less favored compared to other methods.

The chart reveals that Input Capture and OS Credential Dumping are the leading techniques used by malware to gain access to user credentials, highlighting the critical importance of securing input methods and operating system credential storage. The prevalence of methods that exploit unsecured credentials and stored data underscores the need for robust security practices, including the encryption of sensitive information and the use of secure password management tools. Understanding these common credential access techniques is vital for developing effective countermeasures to protect against credential theft and unauthorized access.

6.13. Impact of Multi-Tactic Malware on Detection Rates

The plot in Figure 17 shows how the number of tactics used by malware affects their average detection rate. The x-axis indicates the number of tactics used by a single malware sample, while the y-axis represents the average detection rate. The plot was used to understand how malware complexity in terms of the number of tactics used influences the chances of its detection by security tools.

At low figures there is generally low detection sensitivity; as noted earlier stricter algorithms will always manage to detect any questionable activity regardless of its size (particle-wise) within an instance when looking at these results specifically- close inspection reveals an inverse relationship between our two variables meaning; if one goes up then other goes down this implies that the more action points employed in malware the higher probability it will be caught by protection systems; this might be due to multiple tactics having its ‘digital mark’ on the virus, so there is no way it can be missed. At the lower end, the rate of detection decreases for those instances where there are fewer types of malicious software actions.

Throughout this trend, there are instances of variations with a notable decrease of around 30 tactics before going up again upon completion30. It can be interpreted as changes in terms of how well certain combinations are detected using different specified techniques. The highest detection rates occur when malware makes use of between thirty-five (35) and forty (40) tactics, which shows that while multi-tactic malware may involve greater complexity as well as being more hazardous, it also becomes easier to identify when it employs a broad spectrum.

From the chart, it can be seen if the complexity of malware increases by employing more tactics, then its rate of detection also increases. So that reflected multi-tactic malware, although complex could mean easy detection as it’s seen from far away by detection tools through various activities. Such fluctuations suggest varied efficacies in detection mechanisms based on particular combinations of tactics. Consequently, security measures should address the degree of malware sophistication and type

multiplicity to allow for all-encompassing detection skills necessary for simple and complex threats alike.

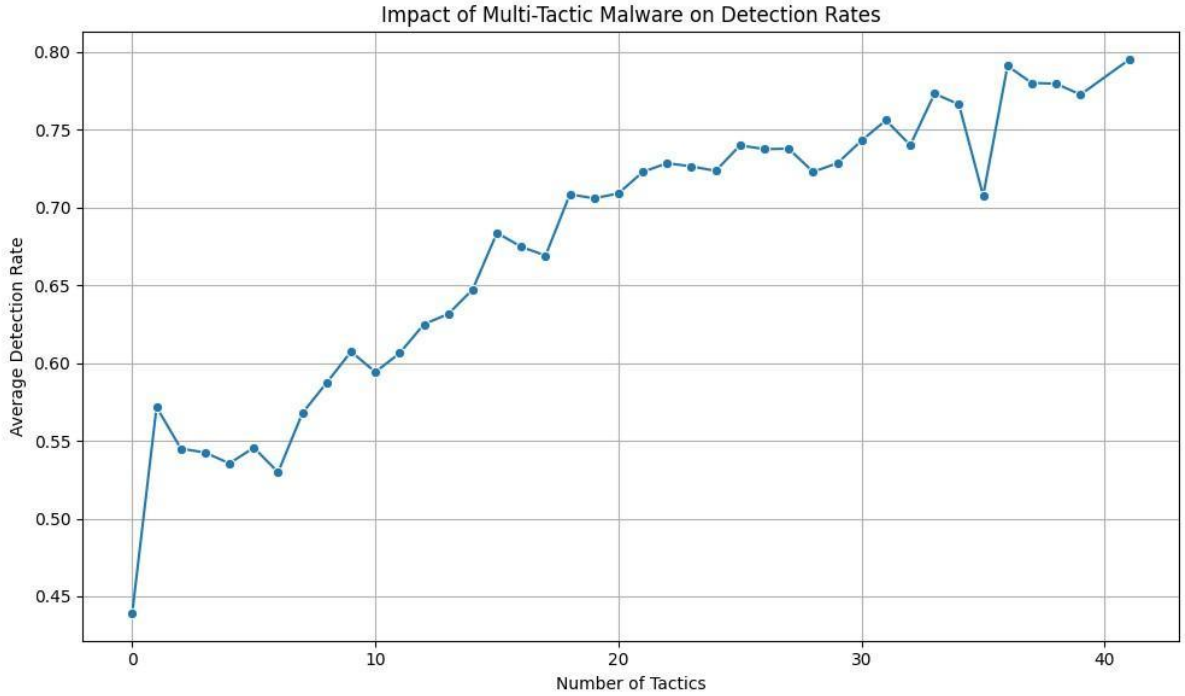

Figure 18 - Impact of Multi-Tactic Malware on Detection Rates

7. Discussion

Thus, the analysis is not without value as it will render some insight into the behaviors, indications, and ways of persistence to show how much problem and difficulty lies before those defending against these threats (cyber cadre).

The popularity graph of persistence methods illustrates that malware chiefly depends upon certain techniques such as Hijack Execution Flow and DLL Side-Loading which are used extensively due to their capability of maintaining presence within compromised systems. These methods have much emphasis on making malware remain strong enough against any attempt to remove them by associating them with ways that systems use like Registry Run Keys as well as Scheduled Task/Job so that they can start each time the computer boots up; thus, ensuring their continuous existence irrespective of efforts made to eliminate them from infected machines. This underscores the importance of elaborate security systems capable of identifying and stopping these kinds of attacks, which are used frequently.

The existence of some malware families like AgentTesla and Formbook which have been found to use a variety of persistence techniques across different malware signatures can be discussed further in light of this distribution. Machines infected with such malware must always stay under their control and, for this reason; they have been designed such that they evade being discovered by antivirus software to avoid creating problems that may impair them…

Comparison is the occurrence of these methods in class A and B types of malware signatures, which helps in understanding how particular strains have been able to evolve. Although certain families may use only a few persistence mechanisms leading to easier identification and subsequent elimination by security programs, there are still aspects of concern based on their specific characteristics.

Another common feature within malware includes combining multiple persistence methods with defense evasion tactics so that the former is enhanced by the latter. Their pairing together makes it very difficult for detection to be done, especially when they merge with others like Virtualization/Sandbox Evasion along with Obfuscated Files. The fact that they use these approaches hints at a necessity, then – we need an approach towards finding such dangerous software that involves looking at it from different perspectives.

But then it was observed through an analysis made on its detection rates that detection of this kind of malware remains difficult through some persistence techniques like Path Interception by Search Order Hijacking while others like Create Account and Change Default File Association often go unnoticed. This is an indication that there is a continuous need for improvement of detection technologies to address mostly less commonly used techniques.

However, if one looks at the Multiple persistence techniques in a single malware sample frequency chart, we see that although most samples do not have numerous persistence techniques, quite a number use more than one such technique to enhance their resilience. This means that even though the majority still relies on single-method persistence mechanisms; there seems to be an increasing trend towards multiple approaches, perhaps for increased chances of remaining undetected. This calls for simple and complex persistent strategies that should be observed by security players.

A comparison of persistence techniques across different signatures suggests that some families have advanced in the use of several persistence methods. In particular, one may argue that AgentTesla and Formbook could be difficult to remove due to the extensive use of persistence techniques as shown above. Such security should be developed and adopted to be able to eliminate multiple persistence methods within a single malware sample.

However, exploring the rarest persistence techniques gives an insight into less frequently used specialized approaches which can still threaten computer systems' security. This might translate to less concern about their detection, as there are hardly any cases where they are used. Consequently, specific malware developers tend to explore some niche fields to avoid detection.

The relationship between persistence and command and control method in use helps to understand how these aspects are interconnected in malware design. As a result, effective detection must not only be based on knowledge about those methods that sustain the malicious software but also about their Control channels.

According to the detection rates across different antivirus vendors, it can be pointed out that in the antivirus market, there are such high performers as Lionic, MAX, and Ikarus. This implies that although there is a minimal gap between them concerning how many numbers of viruses can be detected by either one, continuous innovation and enhancement should keep them at current high levels.

However, the analysis of the prevalence of credential access techniques shows that malware tends to prefer certain methods to others such as Input Capture and OS Credential Dumping for extracting valuable information, hence it brings about numerous challenges in terms of security controls around credentials also imposing data breach risk through unauthorized access to sensitive data.

Finally, examination of multi-tactic malware impact upon detection rates proves that more tactics generally enhance detection frequency, though at times rates fluctuate indicating varying levels of efficacy for different combinations of tactics. This makes it necessary to develop flexible techniques capable of changing concerning the increased complexity of malware, especially when it comes to using a greater number of deceptive modes to avoid being found required for detection purposes.

In conclusion, this discussion focuses on the intricate nature of modern-day malware which often uses a combination of sophisticated persistence, evasion, and credential access methods as a way of surviving.

8. Conclusion and Recommendations

In this thesis, malware persistence techniques have been extensively studied with an emphasis on their prevalence, effect on detection rates, and how they work with different command and control strategies. An analysis found that certain persistence mechanisms for example Hijack Execution Flow and DLL Side-Loading are mostly used by malware since they enable them to remain in a system even after being infected. They help them survive system reboots and thus make detection and removal processes hard. This makes them significant security challenges that must be addressed adequately. The research also concluded that AgentTesla, Formbook, and other malware families use different kinds of chronic methods, making it hard to entirely erase them. Furthermore, some of the malware samples used more than one persistence technique which made them more complicated to detect, hence there is a need for modernized security software capable of recognizing such threats and providing appropriate responses when it comes to defense against software.

The study, however, showed that there exists a relationship between the persistence techniques used by malware and command and control mechanisms that ought to be understood. Such persistence techniques however largely depend on command and control strategies such as execution flow hijacking or DLL manipulation, hence the importance of simultaneously addressing both these aspects through an integrated approach in terms of detection. The research found that when it comes to defense evasion, there are common techniques such as obfuscation or virtualization/sandbox evasion which are most often used by the malware, indicating a clear shift by this group from traditional security tool detections. This shows that there still exists a continuous arms race between malware developers and cybersecurity professionals, with each side constantly evolving its tactics and technologies.

Enhance Detection Capabilities: Continuous improvement in detection technologies is a must, given the sophisticated persistence and evasion techniques employed by modern malware. Security solutions should focus on developing advanced algorithms that can detect and neutralize threats that use multiple persistence methods or sophisticated evasion tactics like obfuscation Integrated Security Solutions: It would be important for cybersecurity frameworks to integrate more detection mechanisms into their systems that can address both persistence and command and control tactics thus making it possible for intercepting any communication between a command center and a malware sample which has managed to establish persistence.

Focus on Common and Rare Techniques Although it is important to defend against commonly used persistence techniques like Hijack Execution Flow, security strategies should also take into account less frequently used but potentially dangerous methods. This comprehensive approach will help in mitigating both widespread and niche threats, thereby enhancing overall system security.

Continuous Monitoring and Adaptation To keep up with the dynamics of malware evolution, organizations need not have static solutions, but adaptive ones. Companies should always have continuous monitoring and regular updates on their security systems to remain relevant in the face of new challenges. This involves being aware of new persistence techniques and incorporating them into your defense strategies.

Education and Awareness For instance add: As malware continues to evolve, cybersecurity experts and users alike must teach them about the latest threats together with the best measures of protecting themselves from such attacks. A regularly updated training program provides an excellent opportunity for organizations to stay updated and well-prepared for potential security breaches

Collaboration and Information Sharing The cybersecurity community should foster partnerships/relationships within organizations, researchers, and security vendors leading to a more robust defense against the changing landscape of threats

In conclusion, the thesis has discussed malware persistence and defense evasion in detail, presenting important information on their distribution and possible solutions. As such, companies will adopt the given suggestions to fortify their systems against modern, complex, and enduring malware threats.

9. Future Work

The research carried out to arrive at this result has shown various forms of malware persistence mechanisms in detail with the use of different tools and approaches. Nonetheless, several directions for further research might develop this study and apply the findings on cybersecurity more broadly.

As one of the future directions of research, it may make sense to expand the existing set of methods based on the analyzing capabilities of the algorithm together with machine learning. With the implication of machine learning in the problem domain, it is possible to achieve predictive modeling for constructing models that warn against new persistence techniques evident in emerging trends and patterns of the historical malware data collected. Such an approach could greatly improve the capability for recognition and counteraction of unseen or emergent risks.

Additionally, machine learning could be leveraged to significantly enhance the classification and characterization of malware based on its persistence mechanisms. By utilizing the extensive dataset collected and analyzed in this study, machine learning algorithms can be trained to identify subtle patterns and behaviors that may not be discernible through manual analysis. These models could offer a more nuanced and precise categorization of malware, allowing for classification not only by the type of persistence technique employed but also by the specific characteristics and traits of the malware itself.

Moreover, machine learning can facilitate the identification of correlations between particular persistence mechanisms and the industries or domains where they are most likely to be deployed. For example, certain persistence techniques might be more prevalent in attacks targeting financial institutions, healthcare organizations, or industrial control systems. By understanding these correlations, machine learning models could predict which sectors are at higher risk of specific types of malware, enabling more targeted and effective defensive strategies.

This capability could prove invaluable for cybersecurity teams, as it would allow them to prioritize resources and tailor their defenses based on the predicted threat landscape. Furthermore, these insights could inform the development of sector-specific training programs for personnel, ensuring that organizations are better equipped to recognize and respond to the most relevant threats in their field. The predictive power of machine learning could also guide investment in security technologies, helping organizations to allocate their budgets more efficiently by focusing on the tools and strategies most likely to counter the threats they face.

Furthermore, the further development of the research could be practiced by integrating the contemporary analysis with threat intelligence platforms. Extending and including current threat intelligence feeds that can be in the form of threat intelligence-sharing communities or idle threat identification systems can improve the study’s ability to produce real-time analytics. This would make it possible to quickly identify new persistence techniques as they are found in the wilds and address them.

Another area for future work, which seems quite promising, is the future studies of cross-platform persistence. Nevertheless, it will be interesting to expand this research focusing on the persistence

mechanisms employed in connection with the malware operating on Linux, macOS, and mobile operating systems. Such a broader view might help to create deeper and more effective detection tools that would shield a greater variety of systems.

Finally, further research could investigate the effectiveness of various defensive tools and strategies against the persistence techniques analyzed in this dissertation. By systematically evaluating the performance of different security solutions, it would be possible to identify gaps in current defenses and recommend specific improvements or new tools that could better address the challenges posed by persistent malware.

In summary, while this dissertation has laid a strong foundation in the analysis of malware persistence techniques, there remains significant potential for further research. By integrating advanced technologies, expanding the scope of analysis, and evaluating defensive measures, future work can continue to enhance the understanding and mitigation of persistent threats in cybersecurity.

10. References

[1] Hama Saeed, Mariwan. (2020). Malware in Computer Systems: Problems and Solutions. IJID (International Journal on Informatics for Development). 9. 1. 10.14421/ijid.2020.09101.

[2] Bettany, Andrew & Halsey, Mike. (2017). What Is Malware?. 10.1007/978-1-4842-2607-0_1.

[3] Herrero-Uribe, Libia. (2011). Viruses, definitions and reality. Revista de biología tropical. 59. 993-8. 10.15517/rbt.v0i0.3372.

[4] Pu, Yiguo & Chen, Xiaojun & Cui, Xu & Shi, Jinqiao & Guo, Li & Qi, Cheng. (2013). Data Stolen Trojan Detection Based on Network Behaviors. Procedia Computer Science. 17. 828-835. 10.1016/j.procs.2013.05.106.

[5] Kiru, Muhammad & Aman, Jantan. (2019). The Age of Ransomware: Understanding Ransomware and Its Countermeasures.. 10.4018/978-1-5225-7353-1.ch001.

[6] Murthy, Rajendran & Balasubramanian, Siva & Hodis, Monica. (2009). Spyware and Adware: How Do Internet Users Defend Themselves?. American Journal of Business. 24. 41-52. 10.1108/19355181200900010.

[7] BELEA, A.-R. (2023). Methods for Detecting Malware Using Static, Dynamic and Hybrid Analysis. *International Conference on Cybersecurity and Cybercrime*, *10*, 258–265. https://doi.org/10.19107/CYBERCON.2023.34

[8] Joy, Jestin & John, Anita & Joy, James. (2011). Rootkit Detection Mechanism: A Survey. 10.1007/978-3-642-24037-9_36.

[9] Zhiqun, Wang & David, Adeyemo & Akinrayo, Akinsoto. (2023). Summary of Cyber Threat Intelligence. 8. 32 - 42. 10.2015/IJIRMF/202203006.

[10] Hassan, Faris & Das, Shampa Rani & Hussain, Manzoor. (2023). Importance of Secure Software Development for the Software Development at Different SDLC Phases. 10.31124/advance.23947392.

[11] Milosevic, Nikola. (2013). History of malware.

[12] Rovic. (2023). Evolution of Malware: A Study Of Evolution of Malware and Its Orgins.

[13] Namanya, Anitta Patience & Cullen, Andrea & Awan, Irfan & Pagna Diss, Jules. (2018). The World of Malware: An Overview. 10.1109/FiCloud.2018.00067.

[14] Sahay, Sanjay & Sharma, Ashu & Rathore, Hemant. (2019). Evolution of Malware and Its Detection Techniques. 10.1007/978-981-13-7166-0_14.

[15] Mallick, Md & Nath, Rishab. (2024). Navigating the Cyber security Landscape: A Comprehensive Review of Cyber-Attacks, Emerging Trends, and Recent Developments.

[16] Gittins, Zane & Soltys, Michael. (2020). Malware Persistence Mechanisms. Procedia Computer Science. 176. 88-97. 10.1016/j.procs.2020.08.010.

[17] Le, Tran Duc & Dinh, Duy & Nguyen T. H., Phuoc & Muthanna, Ammar & Abd El-Latif, Ahmed. (2023). Exploring Common Malware Persistence Techniques on Windows Operating Systems (OS) for Enhanced Cybersecurity Management. 10.1201/9781003369042-7.

[18] Chakkaravarthy, Sibi & Sangeetha, Dimple & Vaidehi, V.. (2019). A Survey on malware analysis and mitigation techniques. Computer Science Review. 32. 1-23. 10.1016/j.cosrev.2019.01.002.

[19] Tahir, Rabia. (2018). A Study on Malware and Malware Detection Techniques. International Journal of Education and Management Engineering. 8. 20-30. 10.5815/ijeme.2018.02.03.

[20] Megira, S & Pangesti, Rizkyana & Wibowo, Ferry. (2018). Malware Analysis and Detection Using Reverse Engineering Technique. Journal of Physics: Conference Series. 1140. 012042. 10.1088/1742-6596/1140/1/012042.

[21] Bhardwaj, Vivek & Kukreja, Vinay & Sharma, Chetan & Kansal, Isha & Popali, Renu. (2022). Reverse Engineering-A Method for Analyzing Malicious Code Behavior. 10.1109/ICAC353642.2021.9697150.

[22] Reddy, Karthik & Bhattacharya, Tathagata & Reddy, Shreevan. (2023). Memory Malware Analysis: Detecting Malicious Signatures In Memory By VolatilityPlugin’s. 10.21203/rs.3.rs-2500418/v1.

[22] Bravo, Pablo & Garcia, Frk Daniel. (2011). Proactive Detection of Kernel-Mode Rootkits. 515 - 520. 10.1109/ARES.2011.78.

[23] Nadim, Mohammad & Lee, Wonjun & Akopian, David. (2023). Kernel-level Rootkit Detection, Prevention and Behavior Profiling: A Taxonomy and Survey.

[24] Tian, Donghai & Ma, Rui & Jia, Xiaoqi & Hu, Changzhen. (2019). A Kernel Rootkit Detection Approach Based on Virtualization and Machine Learning. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2928060.

[25] Beegle, Lynn. (2007). Rootkits and Their Effects on Information Security. Information Systems Security. 16. 164-176. 10.1080/10658980701402049.

[26] Singh, Abhay. (2022). Encrypted Malware Detection Methodology without Decryption using Deep Learning based Approaches. 10.21203/rs.3.rs-2213345/v1.

[27] Brezinski, Kenneth & Ferens, K.. (2021). Metamorphic Malware and Obfuscation -A Survey of Techniques, Variants and Generation Kits. 10.13140/RG.2.2.19702.52802.

[28] Tg, Gregory & Kumar, T.. (2017). A Framework for Dynamic Malware Analysis Based on Behavior Artifacts. 10.1007/978-981-10-3153-3_55.

[29] Khushali, Vala. (2020). A Review on Fileless Malware Analysis Techniques. International Journal of Engineering Research and. V9. 10.17577/IJERTV9IS050068.

[30] Borana, Pramod & Sihag, Vikas & Choudhary, Gaurav & Vardhan, Manu & Singh, Pradeep. (2021). An Assistive Tool For Fileless Malware Detection. 10.23919/WAC50355.2021.9559449.

[31] Jeon, Jaewoo & Cho, Youngho. (2019). Construction and Performance Analysis of Image Steganography-Based Botnet in KakaoTalk Openchat. Computers. 8. 61. 10.3390/computers8030061.

[32] Abhiram, P & Anver, S & Rahiman, M. (2023). A Deep learning framework for domain generation algorithm based malware detection. 10.21203/rs.3.rs-3154412/v1.

[33] Singh, Himanshu Kumar & Singh, Jyoti & Tewari, Anand. (2022). Static Malware Analysis Using Machine and Deep Learning. 10.1007/978-981-19-0604-6_41.

[34] Awang, Norkhushaini & Salleh, Arifin & Darus, Mohamad. (2013). Manual Malware Analysis Using Static Method. International Journal of Computer Networks and Communications Security. 1. 324-328.

[35] Kohli, Navroop & Bindal, Dr. Amit. (2016). A Complete Dynamic Malware Analysis. International Journal of Computer Applications. 135. 20-25. 10.5120/ijca2016908283.

[36] Sihwail, Rami & Omar, Khairuddin & Zainol Ariffin, Khairul Akram. (2018). A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. 8. 1662. 10.18517/ijaseit.8.4-2.6827.

[37] Bazaar Abuse, “Malware Bazaar: A repository of malware samples,” [Online]. Available: https://bazaar.abuse.ch/.

[38] VirusShare, “A collection of malware samples for research purposes,” [Online]. Available: https://virusshare.com/.

[39] Ytisf, “theZoo: A live repository of known malware,” [Online]. Available: https://github.com/ytisf/theZoo.

[40] VirusTotal, “VirusTotal: A tool for analyzing files and URLs for viruses,” [Online]. Available: https://www.virustotal.com/.

[41] MITRE ATT&CK, “A globally-accessible knowledge base of adversary tactics and techniques,” [Online]. Available: https://attack.mitre.org/.

11. Appendices

Scripts:

1. **collect_hashes_mal_bazaar.py**

```
import requests
import os
import re

# Function to fetch all possible tags from MalwareBazaar
def fetch_all_tags_from_malwarebazaar():
    url = "https://mb-api.abuse.ch/api/v1/"
    params = {"query": "get_tags"}
    response = requests.post(url, data=params)
    if response.status_code == 200:
        return [tag['tag'] for tag in response.json().get('data', [])]
    else:
        print(f"Failed to retrieve tags from MalwareBazaar: {response.status_code}")
        return []

# Function to fetch hashes from MalwareBazaar based on a tag
def fetch_hashes_from_malwarebazaar(tag):
    url = "https://mb-api.abuse.ch/api/v1/"
    params = {"query": "get_taginfo", "tag": tag}
    response = requests.post(url, data=params)
    if response.status_code == 200:
        return [sample['sha256_hash'] for sample in response.json().get('data', [])]
    else:
        print(f"Failed to retrieve hashes for tag '{tag}' from MalwareBazaar:
{response.status_code}")
        return []

# Directory to save the hash file
output_dir = '../hashes'
hash_file_path = os.path.join(output_dir, 'all_hashes_mal_baz.txt')

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

def save_hashes_to_file(hashes, file_path):
    with open(file_path, 'a') as f:
        for hash in hashes:
```

```
            f.write(f"{hash}\n")

def remove_duplicate_hashes(file_path):
    with open(file_path, 'r') as f:
            hashes = f.readlines()
    unique_hashes = list(set(hashes))
    with open(file_path, 'w') as f:
            f.writelines(unique_hashes)

if __name__ == "__main__":
    # Fetch all possible tags
    tags = fetch_all_tags_from_malwarebazaar()

    for tag in tags:
            print(f"Collecting hashes for tag: {tag}")
            hashes = fetch_hashes_from_malwarebazaar(tag)
            save_hashes_to_file(hashes, hash_file_path)

    # Remove duplicate hashes
    remove_duplicate_hashes(hash_file_path)
    print(f"Hashes collected and duplicates removed. Final hashes saved in {hash_file_path}")
```

2. **collect_hashes_virus_share.py**

```
import os
import requests

# Directory to save the downloaded hash files
download_dir = '../hashes/virusshare_hashes'

if not os.path.exists(download_dir):
    os.makedirs(download_dir)

# Function to download a file from a URL
def download_file(url, dest_folder):
    if not os.path.exists(dest_folder):
        os.makedirs(dest_folder)

    file_name = os.path.join(dest_folder, url.split('/')[-1])
    print(f"Downloading {url} to {file_name}")
    response = requests.get(url, stream=True)
    if response.status_code == 200:
        with open(file_name, 'wb') as file:
        for chunk in response.iter_content(chunk_size=8192):
        file.write(chunk)
        print(f"Downloaded {file_name}")
    else:
        print(f"Failed to download {url}: {response.status_code}")

# List of base URLs for hash files
base_url = "https://virusshare.com/hashfiles/VirusShare_"
hash_files = [f"{base_url}{str(i).zfill(5)}.md5" for i in range(403, 487)]  # From 403 to 486 (latest hashes 2022 - now)

# Download each hash file
for url in hash_files:
    download_file(url, download_dir)
```

3. **collect_hashes_VT.py**

```
import requests
import os
import time

def load_api_key():
    with open('../config/api_keys.txt', 'r') as file:
        for line in file:
            if line.startswith('VT_API_KEY'):
                return line.strip().split('=')[1]

API_KEY = load_api_key()
HEADERS = {"x-apikey": API_KEY}

REQUEST_RATE_LIMIT = 4  # Maximum requests per minute for free public API
DAILY_QUOTA = 500  # Maximum requests per day for free public API
MONTHLY_QUOTA = 15000  # Maximum requests per month for free public API

def fetch_hashes_from_virustotal(query='malware'):
    url = "https://www.virustotal.com/api/v3/files"
    params = {
        "query": query,
        "limit": 40  # Adjust the limit as needed
    }
    print(f"Fetching hashes with URL: {url}, Headers: {HEADERS}, Params: {params}")
    response = requests.get(url, headers=HEADERS, params=params)
    if response.status_code == 200:
        return [result['id'] for result in response.json().get('data', [])]
    else:
        print(f"Failed to retrieve hashes from VirusTotal: {response.status_code} -
{response.text}")
        return []

def save_hashes_to_file(hashes, file_path):
    with open(file_path, 'a') as f:
        for hash in hashes:
            f.write(f"{hash}\n")

def remove_duplicate_hashes(file_path):
    with open(file_path, 'r') as f:
        hashes = f.readlines()
    unique_hashes = list(set(hashes))
    with open(file_path, 'w') as f:
```

```
            f.writelines(unique_hashes)

def rate_limit(requests_made, start_time):
    # Check if we have reached the request rate limit
    if requests_made >= REQUEST_RATE_LIMIT:
            elapsed_time = time.time() - start_time
            if elapsed_time < 60:
                sleep_time = 60 - elapsed_time
            print(f"Rate limit reached. Sleeping for {sleep_time} seconds.")
            time.sleep(sleep_time)
            return 0, time.time()
    return requests_made, start_time

def log_requests(requests_made_today, requests_made_month):
    with open('../request_log.txt', 'w') as log:
            log.write(f"Requests made today: {requests_made_today}\n")
            log.write(f"Requests made this month: {requests_made_month}\n")

if __name__ == "__main__":
    # Directory to save the hash file
    output_dir = '../hashes'
    hash_file_path = os.path.join(output_dir, 'all_hashes_VT.txt')

    if not os.path.exists(output_dir):
            os.makedirs(output_dir)

    requests_made_today = 0
    requests_made_month = 0
    start_time = time.time()

    # Dummy queries for testing
    queries = ['malware', 'ransomware']
    for i in range(5):  # Limit to 5 fetches
            query = queries[i % len(queries)]
            print(f"Collecting hashes for query: {query}")
            hashes = fetch_hashes_from_virustotal(query)
            save_hashes_to_file(hashes, hash_file_path)

            requests_made_today += 1
            requests_made_month += 1
            requests_made_today, start_time = rate_limit(requests_made_today, start_time)

            if requests_made_today >= DAILY_QUOTA:
            print("Daily quota reached. Stopping for the day.")
```

```
        break

        if requests_made_month >= MONTHLY_QUOTA:
        print("Monthly quota reached. Stopping for the month.")
        break

# Remove duplicate hashes
remove_duplicate_hashes(hash_file_path)
print(f"Hashes collected and duplicates removed. Final hashes saved in {hash_file_path}")

# Log the requests made
log_requests(requests_made_today, requests_made_month)
```

4. **collect_hashes_zoo.py**

```
import os
from git import Repo

# Directory to save the cloned repository and hash file
output_dir = '../thezoo'
hash_file_path = os.path.join(output_dir, 'all_hashes.txt')

# TheZoo repository URL
repo_url = 'https://github.com/ytisf/theZoo.git'

# Function to extract SHA256 hashes from TheZoo's .sha256 files
def extract_hashes_from_thezoo(repo_dir, hash_file_path):
    hashes = []
    sample_dir = os.path.join(repo_dir, 'malware', 'Binaries')
    print(f"Searching for SHA256 hashes in {sample_dir}")

    for root, _, files in os.walk(sample_dir):
        for file in files:
            file_path = os.path.join(root, file)
            if file_path.endswith('.sha256'):
                print(f"Extracting SHA256 hash from file: {file_path}")
                with open(file_path, 'r') as f:
                    hash_value = f.read().strip()
                    hashes.append(hash_value)

    with open(hash_file_path, 'w') as f:
        for hash_value in hashes:
            f.write(f"{hash_value}\n")

    print(f"Extracted {len(hashes)} SHA256 hashes from TheZoo samples.")

if __name__ == "__main__":
    # Clone TheZoo repository
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    repo_dir = os.path.join(output_dir, 'theZoo')
    if not os.path.exists(repo_dir):
        print("Cloning TheZoo repository...")
        Repo.clone_from(repo_url, repo_dir)
    else:
        print("TheZoo repository already cloned.")
```

```
    # Extract hashes from TheZoo samples
    extract_hashes_from_thezoo(repo_dir, hash_file_path)
```

5. collect_malware_info.py

```
import pandas as pd
import json
import sqlalchemy
import pymysql

# Load database credentials from config file
def load_db_credentials(filepath):
    with open(filepath, 'r') as file:
            credentials = {}
            for line in file:
            key, value = line.strip().split('=')
            credentials[key] = value
    return credentials

config = load_db_credentials('../config/db_credentials.txt')

# Database connection
engine                                                                                    =
sqlalchemy.create_engine(f"mysql+pymysql://{config['DB_USER']}:{config['DB_PASS']}@{co
nfig['DB_HOST']}/{config['DB_NAME']}")

# Load data from the database
all_hashes_df = pd.read_sql('SELECT * FROM all_hashes', engine)
malware_info_df = pd.read_sql('SELECT * FROM malware_info', engine)
hash_info_df = pd.read_sql('SELECT * FROM hash_info', engine)

# Helper functions
def extract_detections(detections_json):
    if detections_json is None:
            return {}
    detections = json.loads(detections_json)
    return {av: info['category'] for av, info in detections.items()}

def clean_mitre_techniques(techniques_json):
    if not techniques_json:
            return {}
    techniques = json.loads(techniques_json)
```

```
    if isinstance(techniques, list):
        return {}
    cleaned_techniques = {}
    for source, source_data in techniques.items():
        cleaned_techniques[source] = {
        "tactics": []
        }
        if "tactics" in source_data:
        for tactic in source_data["tactics"]:
        cleaned_tactic = {
            "id": tactic["id"],
            "name": tactic["name"],
            "techniques": []
        }
        for tech in tactic["techniques"]:
            cleaned_tactic["techniques"].append({
            "id": tech["id"],
            "name": tech["name"]
            })
        cleaned_techniques[source]["tactics"].append(cleaned_tactic)
    return cleaned_techniques

def extract_mitre_techniques_names(cleaned_techniques):
    mitre_techs = []
        for source in cleaned_techniques:
        for tactic in cleaned_techniques[source]['tactics']:
        for tech in tactic['techniques']:
           mitre_techs.append(tech['name'])
        return mitre_techs

# Focus on Persistence Techniques
def filter_persistence_techniques(techniques_json):
        cleaned_techniques = clean_mitre_techniques(techniques_json)
        techniques = extract_mitre_techniques_names(cleaned_techniques)
        return [tech for tech in techniques if 'Persistence' in tech]

# Handle None values in 'mitre_att_techniques'
malware_info_df['mitre_att_techniques'] = malware_info_df['mitre_att_techniques'].fillna('[]')

# Extract persistence techniques from the database
malware_info_df['persistence_techniques']                                    =
malware_info_df['mitre_att_techniques'].apply(filter_persistence_techniques)

# Clean detections and mitre_att_techniques
```

```
malware_info_df['detections'] = malware_info_df['detections'].apply(lambda x: json.dumps(extract_detections(x)))
malware_info_df['mitre_att_techniques'] = malware_info_df['mitre_att_techniques'].apply(lambda x: json.dumps(clean_mitre_techniques(x)))

# Merge malware_info_df with hash_info_df to include signatures
malware_info_df = malware_info_df.merge(hash_info_df[['hash', 'signature']], left_on='mal_hash', right_on='hash', how='left')

# Save the prepared data to CSV files
malware_info_df.to_csv('../data/prepared_malware_info.csv', index=False)
detections_df = malware_info_df[['mal_hash', 'detections']].copy()
detections_df['detections'] = detections_df['detections'].apply(json.loads)
detections_df.to_csv('../data/prepared_detections.csv', index=False)

print("Data preparation complete. Data saved to 'prepared_malware_info.csv' and 'prepared_detections.csv'.")
```

6. **combine_hashes.py**

```
import os

# Paths to the input files
file1_path = '../hashes/all_hashes_mal_baz.txt'
file2_path = '../hashes/all_hashes_the_zoo.txt'
output_file_path = '../hashes/combined_hashes.txt'

def combine_and_deduplicate(file1_path, file2_path, output_file_path):
    unique_hashes = set()

    # Read the first file and add hashes to the set
    with open(file1_path, 'r') as file1:
    for line in file1:
    hash_value = line.strip()
    if hash_value:
    unique_hashes.add(hash_value)

    # Read the second file and add hashes to the set
    with open(file2_path, 'r') as file2:
    for line in file2:
    hash_value = line.strip()
    if hash_value:
    unique_hashes.add(hash_value)
```

```
        # Write the unique hashes to the output file
        with open(output_file_path, 'w') as output_file:
        for hash_value in sorted(unique_hashes):
        output_file.write(f"{hash_value}\n")

        print(f"Combined file created at {output_file_path} with {len(unique_hashes)} unique hashes.")

if __name__ == "__main__":
        combine_and_deduplicate(file1_path, file2_path, output_file_path)
```

7. **combine_virusshare_hashes.py**

```
import os

# Directory where hash files are downloaded
download_dir = '../hashes/virusshare_hashes'
combined_file_path = os.path.join(download_dir, 'hashes_VirusShare.txt')

def combine_hash_files(download_dir, combined_file_path):
        unique_hashes = set()
        for root, _, files in os.walk(download_dir):
        for file in files:
        file_path = os.path.join(root, file)
        if file_path.endswith('.md5'):
        with open(file_path, 'r') as hash_file:
                lines = hash_file.readlines()
                  for line in lines:
                  if not line.startswith('#') and line.strip():
                    unique_hashes.add(line.strip())

        with open(combined_file_path, 'w') as combined_file:
        for hash_value in sorted(unique_hashes):
        combined_file.write(f"{hash_value}\n")

        print(f"Combined hash file created at {combined_file_path} with {len(unique_hashes)}
unique hashes.")

if __name__ == "__main__":
        combine_hash_files(download_dir, combined_file_path)
```

8. **data_preparation.py**

```
import pandas as pd
import json
import sqlalchemy
import pymysql

# Load database credentials from config file
def load_db_credentials(filepath):
    with open(filepath, 'r') as file:
        credentials = {}
        for line in file:
        key, value = line.strip().split('=')
        credentials[key] = value
        return credentials

config = load_db_credentials('../config/db_credentials.txt')

# Database connection
engine =
sqlalchemy.create_engine(f"mysql+pymysql://{config['DB_USER']}:{config['DB_PASS']}@{co
nfig['DB_HOST']}/{config['DB_NAME']}")

# Load data from the database
all_hashes_df = pd.read_sql('SELECT * FROM all_hashes', engine)
malware_info_df = pd.read_sql('SELECT * FROM malware_info', engine)
hash_info_df = pd.read_sql('SELECT * FROM hash_info', engine)

# Helper functions
def extract_detections(detections_json):
        if detections_json is None:
        return {}
        detections = json.loads(detections_json)
        return {av: info['category'] for av, info in detections.items()}

def clean_mitre_techniques(techniques_json):
        if not techniques_json:
        return {}
        techniques = json.loads(techniques_json)
        if isinstance(techniques, list):
        return {}
        cleaned_techniques = {}
        for source, source_data in techniques.get('data', {}).items():
        cleaned_techniques[source] = {
```

```
            "tactics": []
            }
            for tactic in source_data.get("tactics", []):
                cleaned_tactic = {
            "id": tactic["id"],
            "name": tactic["name"],
            "techniques": []
            }
            for tech in tactic.get("techniques", []):
              cleaned_tactic["techniques"].append({
                    "id": tech["id"],
                    "name": tech["name"]
            })
           cleaned_techniques[source]["tactics"].append(cleaned_tactic)
            return cleaned_techniques

def extract_mitre_techniques_names(cleaned_techniques):
            mitre_techs = []
            for source in cleaned_techniques:
            for tactic in cleaned_techniques[source]['tactics']:
            for tech in tactic['techniques']:
            mitre_techs.append(tech['name'])
            return mitre_techs

# Handle None values in 'mitre_att_techniques'
malware_info_df['mitre_att_techniques'] = malware_info_df['mitre_att_techniques'].fillna('[]')

# Clean detections and mitre_att_techniques
malware_info_df['detections'] = malware_info_df['detections'].apply(lambda x:
json.dumps(extract_detections(x)))
malware_info_df['mitre_att_techniques'] =
malware_info_df['mitre_att_techniques'].apply(lambda x:
json.dumps(clean_mitre_techniques(x)))

# Merge malware_info_df with hash_info_df to include signatures
malware_info_df = malware_info_df.merge(hash_info_df[['hash', 'signature']],
left_on='mal_hash', right_on='hash', how='left')

# Remove specified columns if they exist
columns_to_remove = ["persistence_techniques", "hash"]
existing_columns = [col for col in columns_to_remove if col in malware_info_df.columns]
malware_info_df = malware_info_df.drop(columns=existing_columns)

# Save the prepared data to CSV files
```

```
malware_info_df.to_csv('../data/prepared_malware_info.csv', index=False)
detections_df = malware_info_df[['mal_hash', 'detections']].copy()
detections_df['detections'] = detections_df['detections'].apply(json.loads)
detections_df.to_csv('../data/prepared_detections.csv', index=False)

print("Data preparation complete. Data saved to 'prepared_malware_info.csv' and 'prepared_detections.csv'.")
```

9. **get_hash_signature.py**

```
import requests
import mysql.connector
import time

# Load configuration from files
def load_config():
    config = {}
    with open('../config/api_keys.txt', 'r') as file:
        for line in file:
        if line.startswith('MALWARE_BAZAAR_API_KEY'):
        config['malware_bazaar'] = line.strip().split('=')[1]
    with open('../config/db_credentials.txt', 'r') as file:
        for line in file:
        key, value = line.strip().split('=')
        config[key] = value
    return config

CONFIG = load_config()
HEADERS_MB = {"API-KEY": CONFIG['malware_bazaar']}
DB_CONFIG = {
    'user': CONFIG['DB_USER'],
    'password': CONFIG['DB_PASS'],
    'host': CONFIG['DB_HOST'],
    'database': CONFIG['DB_NAME']
}

# Function to connect to the database
def connect_db():
    return mysql.connector.connect(**DB_CONFIG)

# Function to query Malware Bazaar for a given hash
def query_malwarebazaar(hash_value, retries=3):
    url = "https://mb-api.abuse.ch/api/v1/"
    data = {
        "query": "get_info",
        "hash": hash_value
    }
    for attempt in range(retries):
        response = requests.post(url, headers=HEADERS_MB, data=data)
        if response.status_code == 200:
        results = response.json()
        if results['query_status'] == 'ok':
```

```
            signature = results['data'][0].get('signature')
            return signature
            elif response.status_code == 502:
            print(f"Server error for {hash_value}, retrying ({attempt + 1}/{retries})...")
            time.sleep(30)
            else:
            print(f"Failed to retrieve details for {hash_value}: {response.status_code} -
{response.text}")
            break
    return None

# Function to insert hash info into the hash_info table
def insert_hash_info(db_conn, hash_value, signature):
    cursor = db_conn.cursor()
    sql = "INSERT INTO hash_info (hash, signature) VALUES (%s, %s)"
    values = (hash_value, signature)
    try:
            cursor.execute(sql, values)
            db_conn.commit()
    except mysql.connector.IntegrityError as e:
            print(f"Hash {hash_value} already exists in the database.")
            except Exception as e:
            print(f"Failed to insert hash {hash_value}: {e}")

# Function to process hashes
def process_hashes(hash_file_path):
            db_conn = connect_db()
            with open(hash_file_path, 'r') as file:
            hashes = file.readlines()
            total_hashes = len(hashes)
              for i, hash_value in enumerate(hashes):
            hash_value = hash_value.strip()
            if hash_value:
            print(f"Processing {i + 1}/{total_hashes}: {hash_value}")
            signature = query_malwarebazaar(hash_value)
            if signature is None:
                signature = "No signature found"
              insert_hash_info(db_conn, hash_value, signature)
            print(f"{hash_value} {signature} +")
            if (i + 1) % 100 == 0:  # Take a break every 100 requests
                        print("Taking a short break to avoid server overload...")
                        time.sleep(60)  # Adjust the sleep time as needed
            db_conn.close()
            print("Processing complete")
```

```
if __name__ == "__main__":
        process_hashes('../hashes/filtered_hashes.txt')
```

10. insert_hashes_to_db.py

```
import mysql.connector
import time

# Load database credentials from configuration files
def load_db_config():
    config = {}
    with open('../config/db_credentials.txt', 'r') as file:
            for line in file:
            key, value = line.strip().split('=')
            config[key] = value
            return config

DB_CONFIG = load_db_config()

# Function to connect to the database
def connect_db():
            return mysql.connector.connect(
            user=DB_CONFIG['DB_USER'],
            password=DB_CONFIG['DB_PASS'],
            host=DB_CONFIG['DB_HOST'],
            database=DB_CONFIG['DB_NAME']
            )

# Function to insert hash into all_hashes table
def insert_hash(db_conn, hash_value):
            cursor = db_conn.cursor()
            sql = "INSERT INTO all_hashes (hash) VALUES (%s)"
            values = (hash_value,)
            try:
            cursor.execute(sql, values)
            db_conn.commit()
            except mysql.connector.IntegrityError as e:
            print(f"Hash {hash_value} already exists in the database.")
            except Exception as e:
            print(f"Failed to insert hash {hash_value}: {e}")

# Function to process hashes
def process_hashes(hash_file_path):
            db_conn = connect_db()
            with open(hash_file_path, 'r') as file:
```

```
        hashes = file.readlines()
        total_hashes = len(hashes)
        for i, hash_value in enumerate(hashes):
        hash_value = hash_value.strip()
        if hash_value:
        print(f"Processing {i + 1}/{total_hashes}: {hash_value}")
        insert_hash(db_conn, hash_value)
        print(f"{hash_value} inserted.")
        if (i + 1) % 100 == 0:  # Take a break every 100 requests
                print("Taking a short break to avoid overloading the database...")
                time.sleep(60)  # Adjust the sleep time as needed
        db_conn.close()
        print("Processing complete")

if __name__ == "__main__":
        process_hashes('../hashes/filtered_hashes.txt')
```

11. **insight_generation.py**

```
import pandas as pd
import json

# Load the data
data_path = "../data/prepared_malware_info.csv"
df = pd.read_csv(data_path)

# Helper function to extract and flatten techniques
def extract_techniques(row, tactic_name):
    try:
        techniques = []
        for engine, data in row.items():
        tactics = data.get('tactics', [])
        for tactic in tactics:
        if tactic['name'] == tactic_name:
                techniques.extend([tech['name'] for tech in tactic['techniques']])
        return list(set(techniques))  # Removing duplicates
        except (KeyError, TypeError, AttributeError):
        return []

# Parse the mitre_att_techniques column properly
df['mitre_att_techniques'] = df['mitre_att_techniques'].apply(lambda x: json.loads(x) if isinstance(x, str) else {})

# Extracting techniques for various tactics
df['persistence_techniques'] = df['mitre_att_techniques'].apply(lambda x: extract_techniques(x, 'Persistence'))
df['defense_evasion_techniques'] = df['mitre_att_techniques'].apply(lambda x: extract_techniques(x, 'Defense Evasion'))
df['command_and_control_techniques'] = df['mitre_att_techniques'].apply(lambda x: extract_techniques(x, 'Command and Control'))
df['credential_access_techniques'] = df['mitre_att_techniques'].apply(lambda x: extract_techniques(x, 'Credential Access'))

# 1. Most Common Persistence Techniques
most_common_persistence = df['persistence_techniques'].explode().value_counts()
most_common_persistence.to_csv('../insights/1.most_common_persistence_techniques.csv')

# 2. Distribution of Persistence Techniques Across Malware Families
persistence_distribution = df.explode('persistence_techniques').groupby(['signature', 'persistence_techniques']).size().reset_index(name='count')
```

```
persistence_distribution.to_csv('../insights/2.persistence_distribution_across_families.csv',
index=False)

# 3. Correlation Between Persistence and Defense Evasion Techniques
persistence_series = df['persistence_techniques'].explode().reset_index(drop=True)
defense_evasion_series = df['defense_evasion_techniques'].explode().reset_index(drop=True)

# Ensure both Series have the same length
min_length = min(len(persistence_series), len(defense_evasion_series))
persistence_series = persistence_series.iloc[:min_length]
defense_evasion_series = defense_evasion_series.iloc[:min_length]

correlation_data = pd.DataFrame({
        'persistence': persistence_series,
        'defense_evasion': defense_evasion_series
}).dropna().reset_index(drop=True)

correlation_counts = correlation_data.groupby(['persistence',
'defense_evasion']).size().reset_index(name='count')
correlation_counts.to_csv('../insights/3.persistence_defense_evasion_correlation.csv',
index=False)

# 4. Unique Persistence Techniques by Malware Signature
unique_persistence = df.groupby('signature')['persistence_techniques'].apply(lambda x:
pd.Series(x.explode().unique())).reset_index(level=0)
unique_persistence.to_csv('../insights/4.unique_persistence_techniques_per_signature.csv',
index=False)

# 5. Impact of Persistence Techniques on Detection Rates
def parse_detections(detections):
        if isinstance(detections, dict):
        return detections
        try:
        return json.loads(detections)
        except (json.JSONDecodeError, TypeError):
        return {}

df['detections'] = df['detections'].apply(parse_detections)
df['av_detection_rate'] = df['detections'].apply(lambda x: sum(1 for val in x.values() if val ==
'malicious') / len(x) if len(x) > 0 else 0)
persistence_impact_on_detection =
df.explode('persistence_techniques').groupby(['persistence_techniques']).agg({'av_detection_rat
e': 'mean'}).reset_index()
```

```
persistence_impact_on_detection.to_csv('../insights/5.persistence_impact_on_detection.csv',
index=False)

# 6. Frequency of Multiple Persistence Techniques in a Single Malware Sample
df['num_persistence_techniques'] = df['persistence_techniques'].apply(len)
multi_persistence_counts = df['num_persistence_techniques'].value_counts().reset_index()
multi_persistence_counts.columns = ['num_persistence_techniques', 'count']
multi_persistence_counts.to_csv('../insights/6.multi_persistence_counts.csv', index=False)

# 7. Comparison of Persistence Techniques Across Different Signatures
persistence_by_signature = df.explode('persistence_techniques').groupby(['signature',
'persistence_techniques']).size().reset_index(name='count')
persistence_by_signature.to_csv('../insights/7.persistence_by_signature.csv', index=False)

# 8. Least Common Persistence Techniques
least_common_persistence = df['persistence_techniques'].explode().value_counts().nsmallest(10)
least_common_persistence.to_csv('../insights/8.least_common_persistence_techniques.csv')

# 9. Relationship Between Persistence Techniques and Command and Control Tactics
persistence_command_control_correlation = pd.DataFrame({
        'persistence': df['persistence_techniques'].explode().reset_index(drop=True),
        'command_control':
df['command_and_control_techniques'].explode().reset_index(drop=True)
}).dropna().reset_index(drop=True)
persistence_command_control_counts =
persistence_command_control_correlation.groupby(['persistence',
'command_control']).size().reset_index(name='count')
persistence_command_control_counts.to_csv('../insights/9.persistence_command_control_correl
ation.csv', index=False)

# Additional Insights
# 1. Detection Rates Across Different Antivirus Vendors
detection_rates_by_av = df['detections'].apply(pd.Series).map(lambda x: 1 if x == 'malicious' else
0).mean().reset_index()
detection_rates_by_av.columns = ['av', 'detection_rate']
detection_rates_by_av.to_csv('../insights/10.detection_rates_by_av.csv', index=False)

# 2. Common Defense Evasion Techniques
common_defense_evasion = df['defense_evasion_techniques'].explode().value_counts()
common_defense_evasion.to_csv('../insights/11.common_defense_evasion_techniques.csv')

# 3. Prevalence of Credential Access Techniques
credential_access_prevalence = df['credential_access_techniques'].explode().value_counts()
credential_access_prevalence.to_csv('../insights/12.credential_access_techniques.csv')
```

```
# 4. Impact of Multi-Tactic Malware on Detection Rates
# Function to count the number of techniques
def count_tactics(mitre_att_techniques):
        count = 0
        for framework, data in mitre_att_techniques.items():
        if isinstance(data, dict):
        tactics = data.get('tactics', [])
        count += len(tactics)
        return count

# Apply the function to the dataframe
df['num_tactics'] = df['mitre_att_techniques'].apply(lambda x: count_tactics(eval(x)) if
isinstance(x, str) else count_tactics(x))
multi_tactic_detection_impact =
df.groupby('num_tactics')['av_detection_rate'].mean().reset_index()
multi_tactic_detection_impact.to_csv('../insights/13.multi_tactic_detection_impact.csv',
index=False)
```

12. plot_creation.py

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os

# Set directory paths
insights_dir = '../insights/'
plots_dir = '../plots/'

# Create plots directory if it does not exist
os.makedirs(plots_dir, exist_ok=True)

# 1. Plot for Most Common Persistence Techniques
most_common_persistence = pd.read_csv(os.path.join(insights_dir, '1.most_common_persistence_techniques.csv'))
most_common_persistence = most_common_persistence.sort_values(by='count', ascending=False).head(15)  # Showing top 15 most common techniques
plt.figure(figsize=(12, 8))
sns.barplot(x=most_common_persistence['count'], y=most_common_persistence['persistence_techniques'], palette='viridis')
plt.title('Most Common Persistence Techniques')
plt.xlabel('Count')
plt.ylabel('Persistence Techniques')
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '1.most_common_persistence_techniques.png'))
plt.close()

# 2. Distribution of Persistence Techniques Across Signatures
# Load the data
persistence_distribution = pd.read_csv(os.path.join(insights_dir, '2.persistence_distribution_across_families.csv'))

# Filter to top 10 most common persistence techniques
top_persistence_techniques = persistence_distribution['persistence_techniques'].value_counts().head(10).index
filtered_data = persistence_distribution[persistence_distribution['persistence_techniques'].isin(top_persistence_techniques)]

# Filter to top 20 signatures based on frequency
top_signatures = filtered_data['signature'].value_counts().head(20).index
filtered_data = filtered_data[filtered_data['signature'].isin(top_signatures)]
```

```
# Create a pivot table for the heatmap
heatmap_data = filtered_data.pivot_table(index='signature', columns='persistence_techniques',
values='count', aggfunc='sum', fill_value=0)

# Plot the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(heatmap_data, annot=True, fmt='d', cmap='Blues', cbar_kws={'label': 'Count'})
plt.title('Top Persistence Techniques Across Selected Signatures')
plt.xlabel('Persistence Techniques')
plt.ylabel('Signatures')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '2.filtered_persistence_distribution_across_families.png'))
plt.close()

# 3. Correlation Between Persistence and Defense Evasion Techniques
# Load the data
persistence_defense_evasion_correlation = pd.read_csv(os.path.join(insights_dir,
'3.persistence_defense_evasion_correlation.csv'))

# Filter to top 10 most common persistence and defense evasion techniques
top_persistence_techniques =
persistence_defense_evasion_correlation['persistence'].value_counts().head(10).index
top_defense_evasion_techniques =
persistence_defense_evasion_correlation['defense_evasion'].value_counts().head(10).index
filtered_data = persistence_defense_evasion_correlation[
    persistence_defense_evasion_correlation['persistence'].isin(top_persistence_techniques) &

persistence_defense_evasion_correlation['defense_evasion'].isin(top_defense_evasion_technique
s)
]

# Create a pivot table for the heatmap
correlation_pivot = filtered_data.pivot_table(index='persistence', columns='defense_evasion',
values='count', aggfunc='sum', fill_value=0)

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_pivot, annot=True, fmt='d', cmap='Blues', cbar_kws={'label': 'Count'})
plt.title('Correlation Between Selected Persistence and Defense Evasion Techniques')
plt.xlabel('Defense Evasion Techniques')
plt.ylabel('Persistence Techniques')
plt.xticks(rotation=45, ha='right')
```

```
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '3.filtered_persistence_defense_evasion_correlation.png'))
plt.close()

# 4. Unique Persistence Techniques by Signature
# Load the data
unique_persistence = pd.read_csv(os.path.join(insights_dir, '4.unique_persistence_techniques_per_signature.csv'))

# Filter to top 20 signatures and top 20 techniques
top_signatures = unique_persistence['signature'].value_counts().head(20).index
top_techniques = unique_persistence['persistence_techniques'].value_counts().head(20).index
filtered_data = unique_persistence[
    (unique_persistence['signature'].isin(top_signatures)) &
    (unique_persistence['persistence_techniques'].isin(top_techniques))
]

# Group and count the occurrences
filtered_data_grouped = filtered_data.groupby(['signature', 'persistence_techniques']).size().reset_index(name='count')

# Create a pivot table for the heatmap
pivot_table = filtered_data_grouped.pivot_table(index='signature', columns='persistence_techniques', values='count', aggfunc='sum', fill_value=0)

# Plot the heatmap
plt.figure(figsize=(14, 12))
sns.heatmap(pivot_table, cmap='YlGnBu', linewidths=.5, annot=True, fmt='.0f', cbar_kws={'label': 'Count'})
plt.title('Top Unique Persistence Techniques by Top Signatures')
plt.xlabel('Persistence Techniques')
plt.ylabel('Signatures')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '4.top_unique_persistence_techniques_per_signature.png'))
plt.close()

# 5. Impact of Persistence Techniques on Detection Rates
# Load the data
persistence_impact = pd.read_csv(os.path.join(insights_dir, '5.persistence_impact_on_detection.csv'))
```

```
# Sort by detection rate and filter top 10 techniques
persistence_impact_sorted = persistence_impact.sort_values(by='av_detection_rate', ascending=False)
top_techniques_impact = persistence_impact_sorted.head(10)

# Plot the top techniques' impact on detection rates
plt.figure(figsize=(12, 8))
sns.barplot(data=top_techniques_impact, x='persistence_techniques', y='av_detection_rate', palette='viridis')
plt.title('Top Persistence Techniques Impact on Detection Rates')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.ylabel('Average Detection Rate')
plt.xlabel('Persistence Techniques')
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '5.top_persistence_impact_on_detection.png'))
plt.close()

# 6. Plot for Frequency of Multiple Persistence Techniques in a Single Malware Sample
multi_persistence_counts = pd.read_csv(os.path.join(insights_dir, '6.multi_persistence_counts.csv'))
plt.figure(figsize=(10, 6))
ax = sns.barplot(x=multi_persistence_counts['num_persistence_techniques'], y=multi_persistence_counts['count'])
ax.set_title('Frequency of Multiple Persistence Techniques in a Single Malware Sample')
ax.set_xlabel('Number of Persistence Techniques')
ax.set_ylabel('Count')
ax.bar_label(ax.containers[0])  # Adding labels on top of the bars
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '6.multi_persistence_counts.png'))
plt.close()

# 7. Comparison of Persistence Techniques Across Different Signatures
# Load the data
persistence_by_signature = pd.read_csv(os.path.join(insights_dir, '7.persistence_by_signature.csv'))

# Filter to top 10 most common persistence techniques
top_persistence_techniques = persistence_by_signature['persistence_techniques'].value_counts().head(10).index
filtered_data = persistence_by_signature[persistence_by_signature['persistence_techniques'].isin(top_persistence_techniques)]

# Select a subset of representative signatures (e.g., top 10 by count)
```

```
top_signatures = filtered_data['signature'].value_counts().head(10).index
filtered_data = filtered_data[filtered_data['signature'].isin(top_signatures)]

# Create a pivot table for the heatmap
pivot_table = filtered_data.pivot_table(index='signature', columns='persistence_techniques', values='count', aggfunc='sum', fill_value=0)

# Plot the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(pivot_table, cmap='YlGnBu', linewidths=.5, annot=True, fmt='.0f', cbar_kws={'label': 'Count'})
plt.title('Comparison of Persistence Techniques Across Selected Signatures')
plt.xlabel('Persistence Techniques')
plt.ylabel('Signatures')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '7.filtered_persistence_by_signature.png'))
plt.close()

# 8. Plot for Least Common Persistence Techniques
least_common_persistence = pd.read_csv(os.path.join(insights_dir, '8.least_common_persistence_techniques.csv'))
least_common_persistence = least_common_persistence.sort_values(by='count', ascending=True).head(10)  # Focusing on the least common 10 techniques
plt.figure(figsize=(10, 6))
sns.barplot(x=least_common_persistence['count'], y=least_common_persistence['persistence_techniques'], palette='viridis')
plt.title('Least Common Persistence Techniques')
plt.xlabel('Count')
plt.ylabel('Persistence Techniques')
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '8.least_common_persistence_techniques.png'))
plt.close()

# 9. Relationship Between Persistence Techniques and Command and Control Tactics
# Load the data
persistence_command_control_correlation = pd.read_csv(os.path.join(insights_dir, '9.persistence_command_control_correlation.csv'))

# Filter to top 10 most common persistence and command_control techniques
top_persistence_techniques = persistence_command_control_correlation['persistence'].value_counts().head(10).index
top_command_control_techniques = persistence_command_control_correlation['command_control'].value_counts().head(10).index
```

```
filtered_data = persistence_command_control_correlation[
   persistence_command_control_correlation['persistence'].isin(top_persistence_techniques) &
         persistence_command_control_correlation['command_control'].isin(top_command_cont
rol_techniques)
]

# Create a pivot table for the heatmap
correlation_pivot = filtered_data.pivot_table(index='persistence', columns='command_control',
values='count', aggfunc='sum', fill_value=0)

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_pivot, annot=True, fmt='d', cmap='Blues', cbar_kws={'label': 'Count'})
plt.title('Relationship Between Selected Persistence Techniques and Command & Control')
plt.xlabel('Command & Control Techniques')
plt.ylabel('Persistence Techniques')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '9.filtered_persistence_command_control_correlation.png'))
plt.close()

# Additional Plots
# 1. Plot for Detection Rates Across Different Antivirus Vendors
detection_rates_by_av = pd.read_csv(os.path.join(insights_dir, '10.detection_rates_by_av.csv'))
detection_rates_by_av = detection_rates_by_av.sort_values(by='detection_rate',
ascending=False).head(20)  # Top 20 AVs for clarity
plt.figure(figsize=(12, 8))
sns.barplot(x=detection_rates_by_av['detection_rate'], y=detection_rates_by_av['av'],
palette='viridis')
plt.title('Top 20 Detection Rates Across Different Antivirus Vendors')
plt.xlabel('Detection Rate')
plt.ylabel('Antivirus Vendor')
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '10.detection_rates_by_av.png'))
plt.close()

# 2. Plot for Common Defense Evasion Techniques
common_defense_evasion = pd.read_csv(os.path.join(insights_dir,
'11.common_defense_evasion_techniques.csv'))
common_defense_evasion = common_defense_evasion.sort_values(by='count',
ascending=False).head(10)  # Showing only top 10 techniques for clarity
plt.figure(figsize=(10, 8))
```

```
sns.barplot(x=common_defense_evasion['count'],
y=common_defense_evasion['defense_evasion_techniques'], palette='viridis')
plt.xticks(rotation=45, ha='right')
plt.title('Top 10 Common Defense Evasion Techniques')
plt.xlabel('Count')
plt.ylabel('Defense Evasion Techniques')
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '11.common_defense_evasion_techniques.png'))
plt.close()

# 3. Plot for Prevalence of Credential Access Techniques
credential_access_prevalence = pd.read_csv(os.path.join(insights_dir,
'12.credential_access_techniques.csv'))
credential_access_prevalence = credential_access_prevalence.sort_values(by='count',
ascending=False).head(10)  # Showing only top 10 techniques for clarity
plt.figure(figsize=(10, 8))
sns.barplot(x=credential_access_prevalence['count'],
y=credential_access_prevalence['credential_access_techniques'], palette='viridis')
plt.xticks(rotation=45, ha='right')
plt.title('Top 10 Prevalence of Credential Access Techniques')
plt.xlabel('Count')
plt.ylabel('Credential Access Techniques')
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '12.credential_access_techniques.png'))
plt.close()

# 4. Plot for Impact of Multi-Tactic Malware on Detection Rates
multi_tactic_detection_impact = pd.read_csv(os.path.join(insights_dir,
'13.multi_tactic_detection_impact.csv'))

# Drop any rows with NaN values if they exist
multi_tactic_detection_impact.dropna(inplace=True)

# Ensure that data is in the correct format
multi_tactic_detection_impact['num_tactics'] =
multi_tactic_detection_impact['num_tactics'].astype(int)
multi_tactic_detection_impact['av_detection_rate'] =
multi_tactic_detection_impact['av_detection_rate'].astype(float)

plt.figure(figsize=(10, 6))
sns.lineplot(data=multi_tactic_detection_impact, x='num_tactics', y='av_detection_rate',
marker='o')
plt.title('Impact of Multi-Tactic Malware on Detection Rates')
plt.xlabel('Number of Tactics')
```

```
plt.ylabel('Average Detection Rate')
plt.grid(True)
plt.tight_layout()
plt.savefig(os.path.join(plots_dir, '13.multi_tactic_detection_impact.png'))
plt.close()

print("Plots have been successfully generated and saved.")
```

13. **reduce_hashes_based_on_date.py**

```
import requests
import os
import time
from datetime import datetime

# Load Malware Bazaar API key from configuration file
def load_api_key():
   with open('../config/api_keys.txt', 'r') as file:
        for line in file:
        if line.startswith('MALWARE_BAZAAR_API_KEY'):
        return line.strip().split('=')[1]

API_KEY = load_api_key()
HEADERS = {"API-KEY": API_KEY}
combined_file_path = '../hashes/combined_hashes.txt'
filtered_file_path = '../hashes/filtered_hashes.txt'
start_date = datetime(2023, 7, 1)

# Function to query Malware Bazaar for a given hash
def query_malwarebazaar(hash_value, retries=3):
        url = "https://mb-api.abuse.ch/api/v1/"
        data = {
        "query": "get_info",
        "hash": hash_value
        }
        for attempt in range(retries):
        response = requests.post(url, headers=HEADERS, data=data)
        if response.status_code == 200:
        results = response.json()
        if results['query_status'] == 'ok':
        date_str = results['data'][0].get('first_seen')
        if date_str:
                date = datetime.strptime(date_str, '%Y-%m-%d %H:%M:%S')
                return date
        elif response.status_code == 502:
        print(f"Server error for {hash_value}, retrying ({attempt + 1}/{retries})...")
        time.sleep(30)
        else:
        print(f"Failed to retrieve details for {hash_value}: {response.status_code} -
{response.text}")
        break
        return None
```

```
# Function to filter hashes by date
def filter_hashes_by_date(combined_file_path, filtered_file_path, start_date):
        with open(combined_file_path, 'r') as file:
        hashes = file.readlines()
        total_hashes = len(hashes)
        with open(filtered_file_path, 'w') as output_file:
        for i, hash_value in enumerate(hashes):
        hash_value = hash_value.strip()
        if hash_value:
                print(f"Processing {i + 1}/{total_hashes}: {hash_value}")
                date = query_malwarebazaar(hash_value)
                if date and date >= start_date:
               output_file.write(f"{hash_value}\n")
        if (i + 1) % 10000 == 0:  # Take a break every 10000 requests
                print("Taking a short break to avoid server overload...")
                time.sleep(60)  # Adjust the sleep time as needed

        print(f"Filtered hashes saved to {filtered_file_path}")

if __name__ == "__main__":
        filter_hashes_by_date(combined_file_path, filtered_file_path, start_date)
```

14. **reform_zoo_hashes.py**

```
import os

# Directory to save the extracted hash files
zoo_dir = '../thezoo'
output_file_path = os.path.join(zoo_dir, 'all_hashes_sha256.txt')

if not os.path.exists(zoo_dir):
        os.makedirs(zoo_dir)

# Function to extract hashes from TheZoo's formatted files
def extract_hashes_from_thezoo(zoo_dir, output_file_path):
        hashes = set()
        for root, _, files in os.walk(zoo_dir):
        for file in files:
        file_path = os.path.join(root, file)
        if file_path.endswith('.txt'):
        with open(file_path, 'r') as hash_file:
                lines = hash_file.readlines()
                for line in lines:
                if ' ' in line:
                    hash_value = line.split(' ')[0]
                  hashes.add(hash_value.strip())

        with open(output_file_path, 'w') as output_file:
        for hash_value in sorted(hashes):
        output_file.write(f"{hash_value}\n")

        print(f"Extracted {len(hashes)} hashes from TheZoo samples and saved to
{output_file_path}")

if __name__ == "__main__":
   extract_hashes_from_thezoo(zoo_dir, output_file_path)
```

15. **rescan__hashes_VT.py**

```
import requests
import mysql.connector
import time
import json

# Load configuration from files
def load_config():
    config = {}
    with open('../config/api_keys.txt', 'r') as file:
            for line in file:
            if line.startswith('VT_API_KEY'):
            config['vt_api_key'] = line.strip().split('=')[1]
    with open('../config/db_credentials.txt', 'r') as file:
            for line in file:
            key, value = line.strip().split('=')
            config[key] = value
    return config

CONFIG = load_config()
HEADERS_VT = {"x-apikey": CONFIG['vt_api_key']}
DB_CONFIG = {
    'user': CONFIG['DB_USER'],
    'password': CONFIG['DB_PASS'],
    'host': CONFIG['DB_HOST'],
    'database': CONFIG['DB_NAME']
}

# Function to connect to the database
def connect_db():
    return mysql.connector.connect(**DB_CONFIG)

# Function to fetch hashes from the all_hashes table between specific limits
def fetch_hashes(db_conn, start_id, end_id):
    cursor = db_conn.cursor()
    query = "SELECT hash FROM all_hashes WHERE id >= %s AND id <= %s"
    cursor.execute(query, (start_id, end_id))
    hashes = [row[0] for row in cursor.fetchall()]
    cursor.close()
    return hashes

# Function to query VirusTotal for a given hash report
def query_virustotal_report(hash_value, retries=3):
```

```
    url = f"https://www.virustotal.com/api/v3/files/{hash_value}"
    for attempt in range(retries):
        response = requests.get(url, headers=HEADERS_VT)
        if response.status_code == 200:
        return response.json()
        elif response.status_code == 502:
        print(f"Server error for {hash_value}, retrying ({attempt + 1}/{retries})...")
        time.sleep(30)
        else:
        print(f"Failed to retrieve report for {hash_value}: {response.status_code} -
{response.text}")
        break
    return None

# Function to query VirusTotal for a given hash MITRE techniques summary
def query_virustotal_mitre(hash_value, retries=3):
    url = f"https://www.virustotal.com/api/v3/files/{hash_value}/behaviour_mitre_trees"
    for attempt in range(retries):
        response = requests.get(url, headers=HEADERS_VT)
        if response.status_code == 200:
        return response.json()
        elif response.status_code == 502:
        print(f"Server error for {hash_value}, retrying ({attempt + 1}/{retries})...")
        time.sleep(30)
        else:
        print(f"Failed to retrieve MITRE techniques for {hash_value}: {response.status_code} -
{response.text}")
        break
        return None

# Function to insert malware info into the malware_info table
def insert_malware_info(db_conn, mal_hash, detections, mitre_att_techniques):
        cursor = db_conn.cursor()
        sql = "INSERT INTO malware_info (mal_hash, detections, mitre_att_techniques)
VALUES (%s, %s, %s)"
        values = (mal_hash, detections, mitre_att_techniques)
        try:
        cursor.execute(sql, values)
        db_conn.commit()
        except mysql.connector.IntegrityError as e:
        print(f"Hash {mal_hash} already exists in the database.")
        except Exception as e:
        print(f"Failed to insert hash {mal_hash}: {e}")
```

```
# Function to process hashes between specific limits
def process_hashes(db_conn, start_id, end_id):
        hashes = fetch_hashes(db_conn, start_id, end_id)
        total_hashes = len(hashes)
        for i, hash_value in enumerate(hashes):
        print(f"Processing {i + 1}/{total_hashes}: {hash_value}")

        # Query VT for the report and MITRE techniques
        report = query_virustotal_report(hash_value)
        mitre = query_virustotal_mitre(hash_value)

        # Extract relevant data
        detections = report['data']['attributes']['last_analysis_results'] if report else None
        mitre_att_techniques = mitre if mitre else None

        # Convert to JSON strings
        detections_json = json.dumps(detections) if detections else None
        mitre_att_techniques_json = json.dumps(mitre_att_techniques) if mitre_att_techniques
else None

        # Insert into the database
        insert_malware_info(db_conn, hash_value, detections_json, mitre_att_techniques_json)

        # Print progress
        print(f"{hash_value} detections and MITRE ATT&CK techniques added to the
database.")

        # Take a short break every 100 requests
        if (i + 1) % 800 == 0:
        print("Taking a short break to avoid server overload...")
        time.sleep(30)

        print("Processing complete")

if __name__ == "__main__":
        db_conn = connect_db()

        # Process hashes between specific limits
        start_id = 2
        end_id = 9500
        process_hashes(db_conn, start_id, end_id)

        db_conn.close()
```