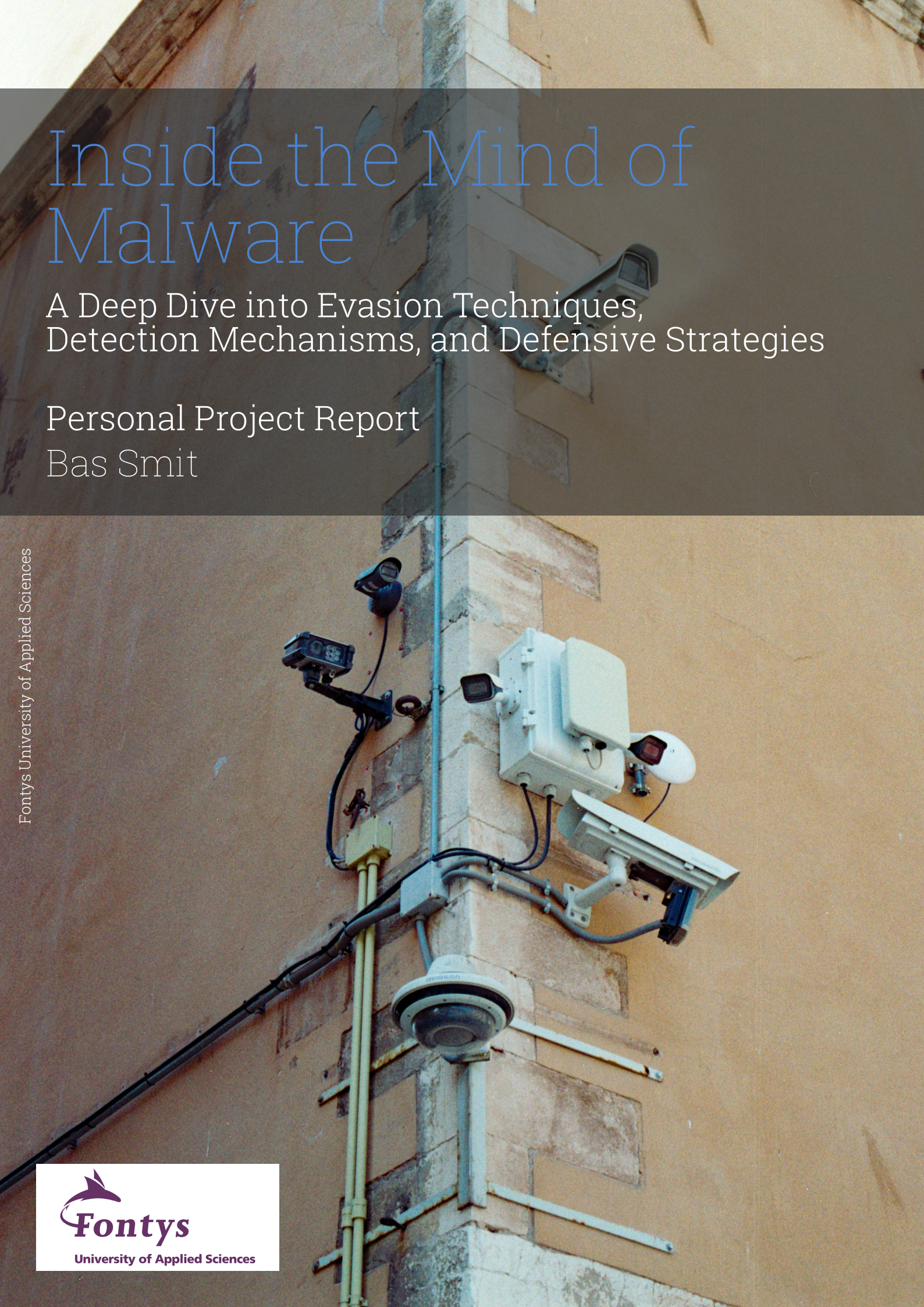
Inside the Mind of Malware

A Deep Dive into Evasion Techniques, Detection Mechanisms, and Defensive Strategies

Personal Project Report

Bas Smit



Fontys University of Applied Sciences

Fontys
University of Applied Sciences

Inside the Mind of Malware

A Deep Dive into Evasion Techniques, Detection Mechanisms, and Defensive Strategies

by

Bas Smit

Instructor:	Schellenkens, Casper C.H.
Teaching Assistant:	Vos, Arno A.F.
Project Duration:	September 2026 - January 2026
Faculty:	Faculty of IT, Eindhoven

Cover:	Security cameras mounted on a building cover by Chris Weiher
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

Fontys HOGESCHOOL ICT

Abstract

The cybersecurity landscape has evolved dramatically from simple proof-of-concept viruses to sophisticated criminal operations targeting financial gain and critical infrastructure. Modern malware employs advanced evasion techniques, including obfuscation, packing, steganography, and binary padding, to bypass traditional security defenses. This research investigates contemporary malware techniques from an offensive security perspective to understand how attackers build, deliver, and maintain malicious software.

Through controlled laboratory experiments, this study examines five key areas: steganographic delivery through malicious PDFs, software packing using UPX, binary padding to evade size-based scanning limits, persistence mechanisms via Windows services, and the effectiveness of signature-based, heuristic-based, and behaviour-based detection methods. The research demonstrates that low-complexity techniques can achieve high-impact results, with traditional signature-based detection proving inadequate against structurally modified malware variants.

Key findings reveal that steganography provides a simple yet powerful delivery mechanism, packing easily defeats signature-based antivirus solutions, and binary padding exploits performance-based scanning weaknesses in commercial security tools. The study concludes that defensive strategies must prioritize behavioural detection, enhance email security and user awareness, implement monitoring of persistence mechanisms, and adopt multi-layered security approaches. These insights provide practical guidance for security analysts, incident responders, SOC teams, and cybersecurity researchers working to strengthen organizational defenses against evolving malware threats.

Keywords: Malware analysis, obfuscation techniques, evasion methods, steganography, software packing, cybersecurity, offensive security, threat detection

Contents

1

Introduction

1.1. Context and Problem Description

The cybersecurity world has changed dramatically over the years. What began with simple viruses that primarily served as proof-of-concept demonstrations has evolved into large-scale criminal operations focused on financial gain and disruption. Early examples like the Creeper worm and the Brain virus were relatively basic and mainly demonstrated how infections could spread. Today, we face ransomware, botnets, phishing campaigns, and other advanced types of malware that are far more complex, automated, and dangerous.

Malware has become a constant and serious threat to both organizations and individuals. It is no longer just about breaking systems. Modern malware is built to steal information, gain unauthorized access, or remain hidden on devices for extended periods. As society becomes increasingly dependent on digital systems for work, communication, and critical infrastructure, attackers have significantly more opportunities to exploit vulnerabilities.

The yearly ENISA Threat Landscape reports clearly illustrate this shift. Phishing has emerged as the primary attack vector, causing approximately 60% of all intrusions. Attackers leverage social engineering, fake emails, and malicious attachments to establish their initial foothold. Simultaneously, ransomware remains the most profitable malware category due to its direct monetization capabilities.

Attackers employ numerous techniques to obfuscate their malware's true functionality. They pack or encrypt code, add padding to executable files, or compile components only after delivery. These methods help bypass traditional antivirus tools. More sophisticated attacks utilize steganography, hiding malicious code inside seemingly benign files such as PDFs or images.

Modern command-and-control (C2) infrastructures make malware even harder to neutralize. While older malware relied on simple centralized servers, contemporary malware families employ peer-to-peer networks, domain generation algorithms, and fast-flux hosting. These techniques significantly complicate takedown efforts.

Because malware continues to evolve, defensive strategies must evolve accordingly. Traditional signature-based antivirus solutions are no longer sufficient. Understanding how attackers build, deliver, obfuscate, execute, and maintain their malware is essential for developing more effective detection and defense mechanisms.

1.2. Research Relevance

This research contributes to the broader cybersecurity field by examining modern malware from an offensive security perspective. By understanding attacker techniques, methodologies, and thought processes, we can provide clearer insights into why certain evasion methods are effective and how defenders can combat them more successfully.

The study benefits multiple stakeholder groups:

- **Security Analysts and Incident Responders** can detect suspicious activity faster and make better decisions during investigations by understanding obfuscation, packing, steganography, and evasion techniques.
- **Security Engineers and SOC Teams** can improve defensive configurations, create better detection rules, and deploy more effective monitoring solutions based on real-world attack patterns.
- **Organizations and Critical Infrastructure Providers** can develop more robust security architectures and implement layered defense strategies that account for modern evasion techniques.
- **Cybersecurity Students and Researchers** gain practical knowledge of offensive techniques in controlled environments, bridging the gap between theoretical concepts and real-world applications.

1.3. Research Scope

This research focuses on practical demonstrations of malware techniques within controlled laboratory environments. The study examines steganographic delivery, software packing, binary padding, persistence mechanisms, and various detection methodologies. All experiments are conducted in isolated virtual machines with no connection to production networks, ensuring ethical research practices.

The research deliberately excludes real malware deployment, production C2 infrastructure, and attacks on live systems. Instead, it uses custom-developed proof-of-concept implementations to illustrate technical concepts safely and responsibly.

2

Theoretical and Practical Background

This chapter establishes the foundational knowledge necessary to understand modern malware threats, detection mechanisms, and evasion techniques. It provides context for the research questions and methodology presented in subsequent chapters.

2.1. What is Malware?

Malware, short for malicious software, refers to any software intentionally designed to cause damage to a computer, server, client, or computer network. Unlike software bugs, which are unintentional flaws, malware is deliberately created with malicious intent to infiltrate systems, steal sensitive data, disrupt operations, or gain unauthorized access. The distinction between malware and legitimate software lies in its purpose and the consent of the end user.

2.2. Types of Malware

Understanding the taxonomy of malware is essential for analyzing detection and evasion strategies. Different malware types employ distinct techniques and target specific attack vectors.

2.2.1. Viruses

Computer viruses are self-replicating programs that attach themselves to legitimate executables or documents. They propagate by executing when the host program runs, infecting additional files on the system and potentially spreading to other systems through shared resources or removable media.

2.2.2. Trojans

Trojan horses disguise themselves as legitimate software while concealing malicious functionality. Unlike viruses, Trojans do not self-replicate but rely on social engineering to trick users into executing them. Common examples include fake system utilities, pirated software, and backdoored applications.

2.2.3. Ransomware

Ransomware encrypts victim data using strong cryptographic algorithms and demands payment, typically in cryptocurrency, for decryption keys. Modern ransomware variants often employ double-extortion tactics, threatening to publicly release stolen data alongside encryption, significantly increasing pressure on victims.

2.2.4. Botnets

Botnets consist of networks of compromised devices (bots or zombies) controlled remotely by attackers through command-and-control infrastructure. These networks enable coordinated attacks including distributed denial-of-service (DDoS), spam distribution, credential stuffing, and cryptomining operations.

2.3. Malware Prevalence and Attack Vectors

Understanding current threat trends is crucial for prioritizing defensive strategies and research efforts. According to European Union Agency for Cybersecurity (2025), phishing accounts for approximately 60% of initial access vectors, leveraging social engineering rather than technical vulnerabilities. Ransomware remains the most financially lucrative malware category, with average ransom demands exceeding millions of dollars for enterprise targets. Additionally, supply chain attacks have emerged as a sophisticated threat vector, compromising legitimate software distribution channels to achieve widespread impact with minimal effort.

2.4. Obfuscation and Evasion Techniques

Obfuscation and evasion techniques play a central role in modern malware development by enabling malicious code to avoid detection and hinder analysis. As detection mechanisms improve, attackers continuously adapt these techniques to remain effective. This section expands on the primary obfuscation methods examined in this research and illustrates how they operate in practice.

2.4.1. Steganography

Steganography is the practice of concealing malicious code or data within benign looking carrier files such as images, PDF documents, audio files, or network traffic. Unlike encryption, which makes data unreadable but clearly signals the presence of protected content, steganography aims to hide the existence of the payload entirely. As a result, steganographic malware often bypasses security controls that rely on file type trust or superficial inspection.

How it works:

- **Least significant bit manipulation:** Individual bits in image or audio data are modified in a way that does not noticeably affect visual or audio quality.
- **Abuse of metadata fields:** Malicious data is stored in optional metadata sections that are rarely validated or displayed to users.
- **Format exploitation:** Legitimate file format features are abused to embed executables or shellcode within complex structures such as PDFs.

Example 1: Malicious payload embedded in a PDF. A PDF file may appear to contain only text or images but internally includes an embedded executable or script object. When opened, the document triggers vulnerable functionality or scripting logic that extracts and executes the hidden payload, establishing a reverse shell or downloading additional components.

Example 2: Image based steganography. In an image file, the least significant bits of pixel values can be altered to encode binary data. To a user and basic scanners, the image appears unchanged. At runtime, a loader extracts the hidden data, reconstructs the payload in memory, and executes it. This allows malware delivery through platforms that only permit image uploads.

Evasion characteristics:

- Exploits implicit trust in common file formats
- Bypasses content filters that focus on file extensions
- Avoids obvious indicators such as encrypted blobs or executable headers

2.4.2. Software Packing

Software packing transforms an executable by compressing or encrypting its contents and wrapping it in a small loader stub. When executed, the stub dynamically unpacks the original program into memory and transfers execution to it. This prevents static analysis tools from inspecting the original code without executing or manually unpacking the binary.

How it works:

- The original executable is compressed or encrypted
- A loader stub is prepended to the file

- At runtime, the stub allocates memory, restores the original code, and jumps to its entry point

Example 1: Single layer packing with UPX. When an executable is packed with UPX, its file size and checksum change significantly. Static scanners relying on known signatures fail to match the packed version, even though the unpacked program behaves identically at runtime. UPX can also remove or minimize import tables, further hindering static inspection.

Example 2: Multi layer and polymorphic packing. Advanced packers apply multiple layers of packing, where each unpacking stage reveals another packed layer. Polymorphic packers further randomize encryption keys, loader logic, and section layouts for each build. This results in binaries that are functionally identical but structurally unique, making signature reuse ineffective.

Impact on analysis:

- Breaks hash based and byte signature detection
- Obscures control flow and strings
- Forces analysts to rely on dynamic analysis or manual unpacking

2.4.3. Binary Padding

Binary padding is a simple but effective evasion technique that inflates the size of an executable by appending non functional data, typically null bytes, to the end of the file. This padding does not affect program execution but can influence how security tools prioritize or process the file.

How it works:

- Large quantities of filler data are appended beyond the valid PE sections
- The operating system ignores this overlay data during execution
- Antivirus engines must process significantly larger files during scanning

Example: Size based scanning evasion. Some antivirus engines impose file size thresholds to reduce scanning overhead. When an executable is padded to hundreds of megabytes, the scanner may skip deep inspection, apply reduced heuristics, or time out during analysis. Despite the inflated size, the malicious code remains small and executes normally because only the defined executable sections are loaded into memory.

Evasion characteristics:

- Changes file hashes and digital fingerprints
- Exploits performance and timeout limitations in scanners
- Can be combined with packing for compounded evasion effects

2.4.4. Combined Evasion Strategies

In real world malware, obfuscation techniques are rarely used in isolation. Attackers often combine steganography for delivery, packing for static analysis evasion, and binary padding for scanner avoidance. This layered approach increases resilience against individual detection mechanisms and forces defenders to rely on multi stage analysis pipelines rather than single technique detection.

2.5. Malware Lifecycle and Attack Chain

Understanding the malware lifecycle provides insight into defensive opportunities at each stage. The lifecycle, often mapped to frameworks like MITRE ATT&CK, consists of several phases:

1. **Initial Access** – Gaining entry into target systems through vectors such as phishing emails, drive-by downloads, software vulnerabilities, or compromised credentials
2. **Execution** – Running malicious code on the target system, often leveraging scripting engines, exploitation frameworks, or user-initiated actions
3. **Persistence** – Establishing mechanisms to maintain access across system reboots and updates through registry modifications, scheduled tasks, or service installations

4. **Privilege Escalation** – Exploiting vulnerabilities or misconfigurations to obtain higher-level system permissions, enabling deeper system access
5. **Defense Evasion** – Employing techniques to avoid detection by security tools, including process injection, rootkits, and disabling security software
6. **Command and Control (C2)** – Establishing communication channels with attacker infrastructure for receiving commands and transmitting stolen data
7. **Exfiltration** – Stealing sensitive data from compromised systems through encrypted channels or legitimate services to avoid detection

Each phase presents opportunities for detection and intervention, making understanding this progression essential for effective defense strategies.

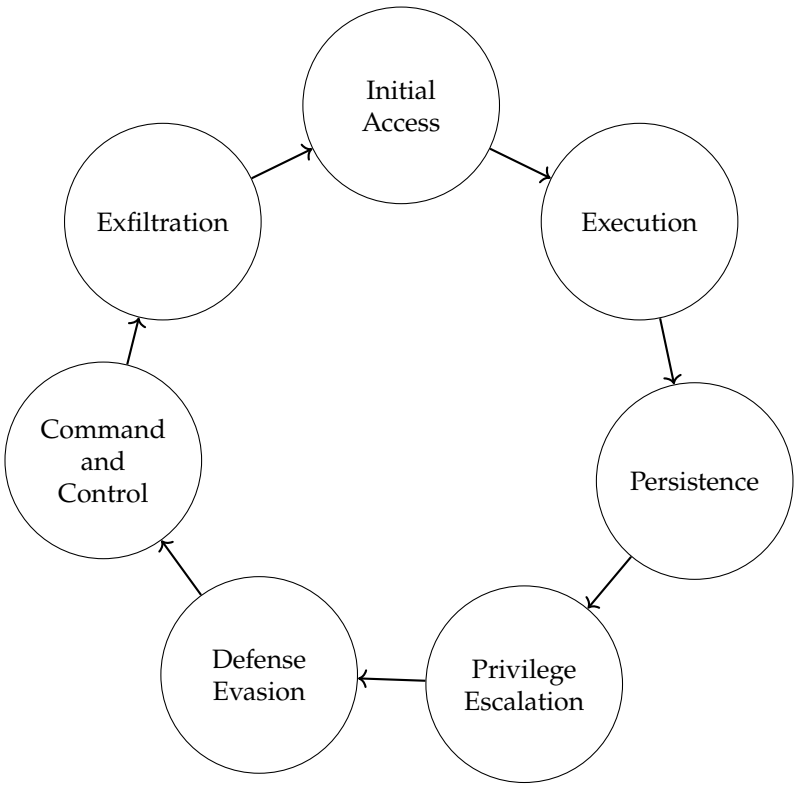


Figure 2.1: Circular malware lifecycle

2.6. Command-and-Control (C2) Architectures

Command-and-control infrastructure enables attackers to maintain communication with compromised systems. C2 architecture design significantly impacts both operational resilience and detection difficulty.

2.6.1. Centralized C2

Traditional C2 infrastructure relies on centralized servers or domains for command distribution and data reception. While simpler to implement and manage, centralized architectures create single points of failure, making them vulnerable to identification, monitoring, and takedown by law enforcement or security researchers.

2.6.2. Decentralized and Resilient C2

Modern malware increasingly employs sophisticated techniques to create resilient communication channels. Peer-to-peer (P2P) networks distribute C2 functionality across infected machines, eliminating single points of failure. Domain Generation Algorithms (DGAs) dynamically create large numbers of potential C2 domains, requiring defenders to predict and block hundreds or thousands of domains.

Fast-flux DNS rapidly rotates IP addresses associated with C2 domains, complicating blocking efforts. Additionally, some advanced threats leverage legitimate services such as cloud storage, social media platforms, or public paste sites as covert communication channels, blending malicious traffic with normal user activity.

2.7. Detection Methods

Effective malware detection relies on multiple complementary approaches, each targeting different stages of the attack lifecycle. Modern endpoint security solutions typically apply layered detection pipelines that combine static indicators (signatures), structural inference (heuristics), and runtime telemetry (behavior). This section expands the three core detection methods used in this research and provides concrete examples of how each method works in practice.

2.7.1. Signature-Based Detection

Signature-based detection identifies malware by matching known indicators against curated databases. These indicators can include cryptographic hashes (e.g., MD5, SHA-256), fixed byte sequences, file headers, embedded strings, or rule-based patterns. The primary advantage of this approach is precision: when a signature matches, the detection is typically highly reliable and fast, with low computational cost. However, signature-based systems are inherently reactive, and even small modifications can invalidate a signature.

How it works:

- **Hash matching:** The scanner computes a file hash and compares it against a blacklist of known malicious hashes.
- **Byte pattern matching:** The scanner searches for specific byte sequences at known offsets or anywhere in the file.
- **Rule matching:** Signature rules (often similar to YARA logic) combine multiple conditions such as strings, byte patterns, and file structure checks.

Example 1: Hash-based detection failure after packing. If an executable is detected using a SHA-256 hash, compressing or packing the file with a tool such as UPX changes the resulting file bytes. Even if the program behavior is identical, the hash becomes different, and hash-based signature detection fails because the file no longer matches a stored fingerprint.

Example 2: Byte signature evasion through minor edits. Assume an antivirus signature detects a specific byte sequence in a payload stub. If the attacker inserts a few no-op instructions, reorders functions, changes compiler flags, or adds padding, the byte sequence changes enough to break the match. This is the core reason why polymorphism and small code changes often defeat signature-only solutions.

Limitations:

- Ineffective against new malware families and zero-day payloads
- Fragile against minor modifications such as packing, re-compilation, and padding
- Requires continuous signature updates and threat intelligence feeds

2.7.2. Heuristic-Based Detection

Heuristic-based detection attempts to identify malware by analyzing suspicious characteristics rather than exact known signatures. This can include static heuristics (without running the code) and emulation-based heuristics (partial execution or instruction tracing). Heuristics are designed to generalize, making them useful for detecting previously unseen malware variants that share common structural traits.

How it works:

- **Static heuristics:** Inspect file structure, import tables, section permissions, entropy, suspicious strings, or unusual metadata.

- **API and capability inference:** Detect risky combinations such as network APIs plus process injection APIs.
- **Emulation heuristics:** Execute code in a lightweight emulator for a short time window and flag suspicious instruction sequences.

Example 1: Suspicious API combination. A benign calculator program typically imports common libraries for GUI or arithmetic. In contrast, a suspicious sample may import APIs such as:

- `VirtualAlloc` or `VirtualProtect` (memory allocation and permission changes)
- `CreateRemoteThread` (thread injection into another process)
- `WinHttpOpen` or `WSAStartup` (network communication)

Heuristic logic may flag the combination of memory manipulation APIs with network APIs as high risk because this pattern commonly appears in droppers and shellcode loaders.

Example 2: High entropy and packed file indicators. Packed binaries often exhibit high entropy sections due to compression or encryption-like byte distributions. A heuristic detector may compute entropy values and flag samples where:

- the `.text` section entropy is unusually high,
- section names are abnormal or generic,
- the import table is minimal (common in packed samples),
- the entry point is located in an unexpected section.

This does not confirm malware by itself, but it increases suspicion and may trigger deeper analysis.

Limitations:

- Higher false positive rates, especially for uncommon but legitimate software (e.g., installers, admin tools)
- Susceptible to evasion through obfuscation, string encryption, and import hiding
- Requires careful tuning to avoid overly broad detections

2.7.3. Behavior-Based Detection

Behavior-based detection, also known as dynamic analysis, focuses on what a program does during execution rather than how it looks statically. This approach monitors runtime telemetry including process creation, file operations, registry modifications, network activity, credential access, and privilege escalation attempts. Because it evaluates actions, it is effective against polymorphic malware and many zero-day threats that evade signatures and static heuristics.

How it works:

- **Runtime monitoring:** The endpoint agent collects events (process, registry, filesystem, network).
- **Behavior rules:** Rules detect suspicious sequences, such as a document process spawning a shell.
- **Correlation and scoring:** Multiple low-confidence actions are combined into a higher-confidence detection.
- **Sandbox execution:** Suspicious files may be detonated in isolated environments to observe behavior safely.

Example 1: Reverse shell behavioral indicators. A reverse shell demonstration typically shows patterns such as:

- a process initiates an outbound connection to an unexpected IP and port,
- a command interpreter (e.g., `cmd.exe` or `powershell.exe`) starts without user interaction,
- the interpreter reads commands from a network socket and executes them.

Even if the payload is packed or obfuscated, these runtime actions remain observable and can trigger behavioral alerts.

Example 2: Persistence through registry Run keys. A common persistence technique is writing a startup entry under:

- `HKCU:\Software\Microsoft\Windows\CurrentVersion\Run`

Behavior-based detection can flag:

- creation or modification of Run key values,
- suspicious naming that mimics system components,
- immediate execution of a hidden script after persistence is set,
- repeated execution across logins or reboots.

Example 3: Document-based infection chain. If a PDF or Office document process spawns a script engine or shell (e.g., `AcroRd32.exe` launching `powershell.exe`), a behavior-based detector may classify this as highly suspicious because it matches a common malware delivery chain: document open -> script execution -> payload download or execution.

Limitations:

- Requires execution or observation time, increasing analysis cost
- Sandbox environments can be detected by malware through timing checks or environment fingerprinting
- Delayed execution and user-interaction triggers may reduce visibility in short sandbox windows

2.7.4. Layered Detection in Practice

In operational security products, these methods are rarely used in isolation. A typical layered workflow is:

1. **Signature scan** for fast elimination of known threats
2. **Heuristic analysis** to identify suspicious characteristics and prioritize deeper inspection
3. **Behavior monitoring** or **sandbox detonation** to confirm malicious intent through runtime evidence

This layered approach reduces false negatives and increases resilience against evasion techniques such as packing, padding, and code obfuscation, while still maintaining acceptable performance and detection accuracy.

3

Research Questions

3.1. Main Research Question

How do modern malware techniques evade detection, and what defensive strategies can effectively counter these evasion methods?

3.2. Sub-questions

To address the main research question comprehensively, this study investigates the following sub-questions:

1. **How effective is steganography as a malware delivery mechanism?**

   This question examines the practical implementation of steganographic techniques for hiding malicious payloads within benign file formats such as PDFs, and assesses whether this method successfully evades traditional security controls.

2. **Can software packing bypass signature-based antivirus detection?**

   This investigates whether packing executables with tools like UPX can defeat signature-based detection systems, and explores the limitations of static analysis approaches.

3. **Does binary padding exploit performance-based scanning limitations?**

   This question tests whether artificially inflating file sizes can cause antivirus software to skip scanning due to performance constraints, effectively bypassing security checks.

4. **How easily can malware establish persistence on Windows systems?**

   This examines the implementation of persistence mechanisms through Windows services and evaluates how trivial it is for attackers to maintain long-term access.

5. **What are the relative effectiveness of signature-based, heuristic-based, and behaviour-based detection methods?**

   This compares different detection methodologies to determine which approaches provide the most robust protection against modern evasion techniques.

6. **What defensive strategies should organizations prioritize?**

   Based on experimental findings, this question identifies practical recommendations for security teams to strengthen their defensive posture against contemporary malware threats.

3.3. Research Objectives

The research aims to:

- Demonstrate practical implementation of common malware evasion techniques in controlled environments
- Evaluate the effectiveness of different detection methodologies against obfuscated malware
- Provide actionable insights for security professionals to improve organizational defenses
- Bridge the gap between theoretical knowledge and practical offensive security understanding
- Contribute to the body of knowledge on malware analysis and detection strategies

4

Methodology

4.1. Research Approach

For this research I used a hands-on experimental approach where I set up a controlled lab environment to test different malware techniques. The methodology is based on the DOT (Development Oriented Triangulation) framework Egele et al., 2012; Sikorski and Honig, 2012, which basically means I combined three things: reading up on existing literature about malware, doing practical experiments in virtual machines, and analyzing real-world data about malware detection.

I started by reviewing academic papers and security reports to understand what techniques are commonly used by malware authors European Union Agency for Cybersecurity, 2025; Symantec, 2020; You and Yim, 2010. Then I recreated some of these techniques in my lab to see how they actually work and whether they can bypass antivirus software. Finally, I looked at real-world statistics to see which detection methods are most effective.

4.2. Tools and Environment

All experiments were done in isolated Windows 10 virtual machines to make sure nothing could escape and affect my actual computer Sikorski and Honig, 2012. I disabled external networking on the VMs so they couldn't communicate outside the lab. For the experiments I used several tools:

- Metasploit Framework for generating payloads
- Apache HTTP Server to host malicious files
- UPX for packing executables Ugarte-Pedrero et al., 2015; You and Yim, 2010
- Python with PyInstaller for creating custom executables
- A custom antivirus simulator I built to test detection methods Egele et al., 2012

The custom antivirus simulator was particularly important because it let me test how signature-based, heuristic-based, and behaviour-based detection would respond to the different evasion techniques.

4.3. Experiments

I ran several experiments to test different malware techniques and see how well they can evade detection. Each experiment focused on a specific technique that's commonly used by real malware. Here's what I did:

Experiment 1: Hiding Malware in PDFs

The goal here was to show how attackers can embed malware inside a seemingly innocent PDF file. I used Metasploit to create a PDF that looks normal but contains a reverse shell payload. When someone opens the PDF, it secretly connects back to the attacker's machine without the user noticing anything suspicious.

I set up a web server to host the malicious PDF (in real attacks this would be sent via phishing emails). Then I configured Metasploit's `exploit/fileformat/adobe_pdf_embedded_exe` module with my listener's IP and port. When the PDF is opened, the embedded shellcode connects back to the attacker machine.

Figure 4.1: Creating the malicious PDF payload in Metasploit

On the Windows VM, when I opened the PDF it looked like a normal document. But in the background, the payload executed and I got a reverse shell connection on my Metasploit listener (Figure 4.3). From there I could run commands on the victim machine and explore the file system. This shows how dangerous seemingly harmless files can be.

Figure 4.2: The malicious PDF hosted on a local web server

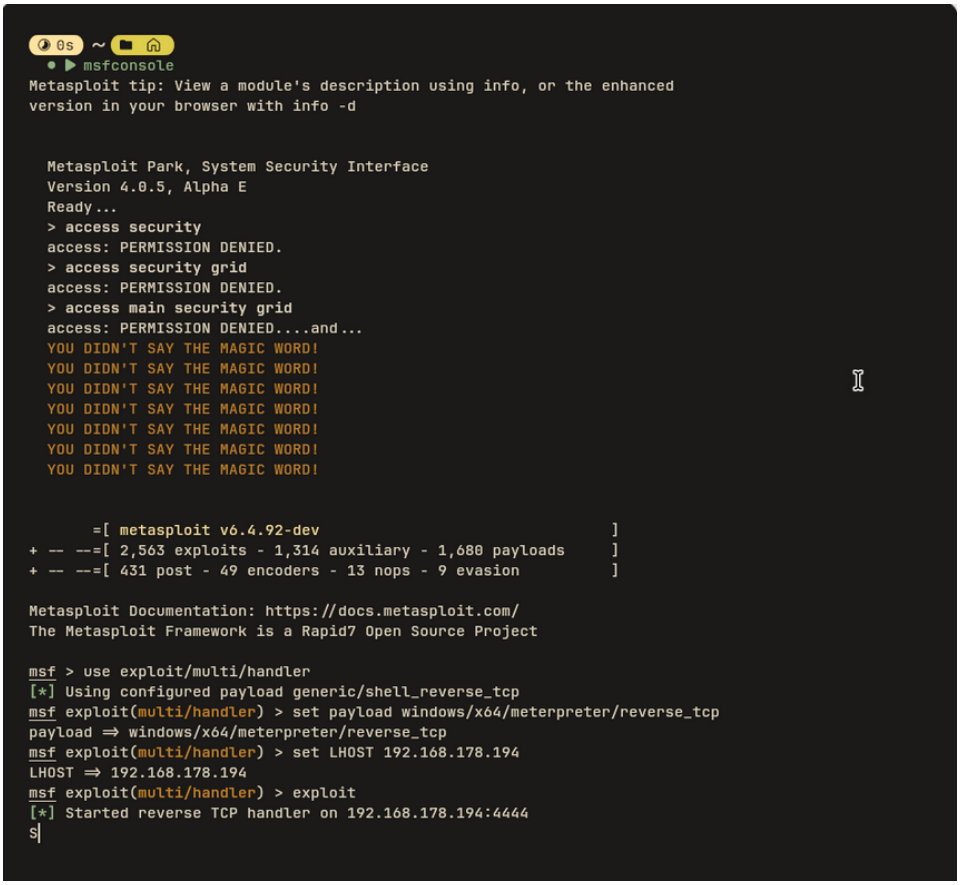


Figure 4.3: Successful reverse shell after opening the PDF

Experiment 2: Software Packing with UPX

For this experiment I wanted to see how packing an executable changes its signature while keeping it

functional. I wrote a simple C program that just prints "Hello" to the console, then compiled it with gcc. After checking the original file's size and hash, I used UPX to pack it.

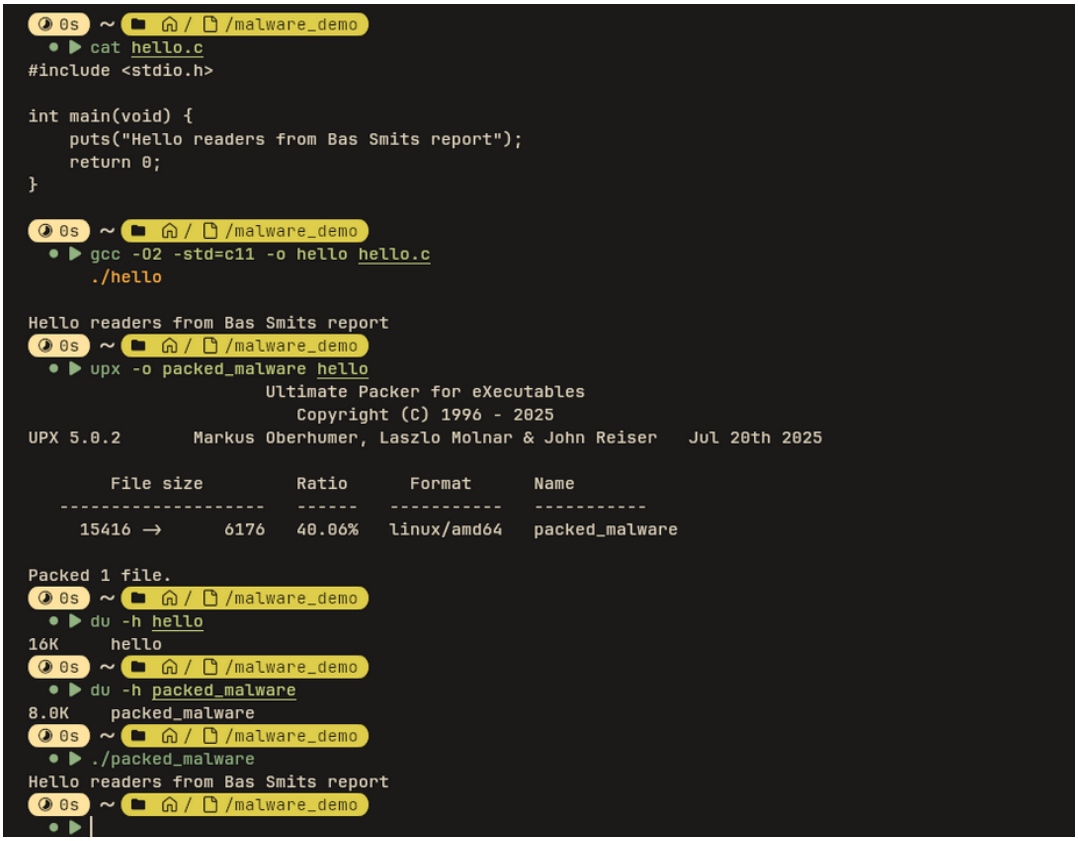


Figure 4.4: Using UPX to pack the executable

The results were interesting - the file size decreased significantly and more importantly, the hash completely changed (Figure 4.5). This means any signature-based antivirus that only knows the original hash won't recognize the packed version. When I ran the packed executable, it still worked exactly the same as before, printing "Hello" just like the original.

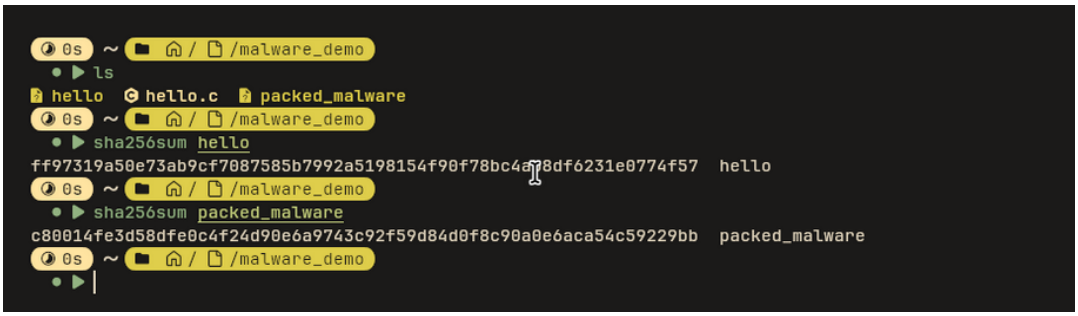


Figure 4.5: Different hashes before and after packing

Experiment 3: Binary Padding

This technique is pretty clever - you just add massive amounts of junk data to the end of an executable to make it huge. I learned about this from a real Emotet malware campaign where they padded files to over 500 MB Trend Micro Research, 2023. The crazy thing is these files could be compressed to less than 1 MB, proving almost all of it was just padding.

I wrote a Python script that appends null bytes (0x00) to executables. I tested different sizes - 50MB, 100MB, and 500MB - to see how antivirus scanners would react. The idea is that many scanners have timeout limits or file size restrictions, so they might skip scanning huge files or only scan part of them.

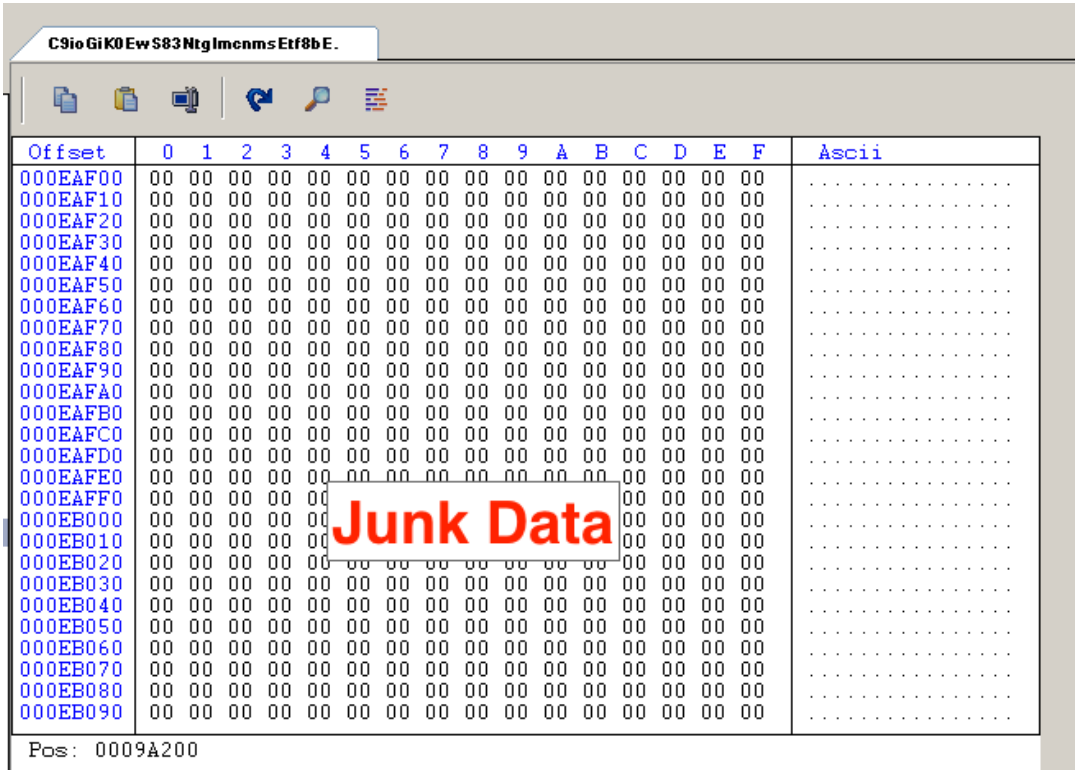


Figure 4.6: Python script adding junk data to the executable

```
[Sections]

nth paddr          size vaddr         vsize perm name
-------------------------------------------------------
0   0x00000800  0x4d400 0x00401000  0x4e000 -r-x .text
1   0x0004dc00   0x3600 0x0044f000   0x4000 -rw- .rodata
2   0x00051200   0xe400 0x00453000  0x23000 -rw- .data
3   0x0005f600    0x600 0x00476000   0x1000 -rw- .tls
4   0x0005fc00   0x4600 0x00477000   0x5000 -r-- .pdata
5   0x00064200   0x6800 0x0047c000   0x7000 -r-- .xdata
6   0x0006aa00    0xc00 0x00483000   0x1000 -r-- .idata
7   0x0006b600   0x1c00 0x00484000   0x2000 -r-- .edata
8   0x0006d200  0x2be00 0x00486000  0x2c000 -r-- .rsrc
9   0x00099000   0x1200 0x004b2000   0x2000 sr-- .reloc
```

Figure 4.7: File structure showing where the padding is added

The interesting thing is that when Windows runs the padded executable, it only loads the actual PE

sections into memory and ignores all the padding. So the program still works perfectly while appearing massive on disk.

Experiment 4: Windows Service Persistence

For persistence, I first tried using Windows services. I wrote a Python script that creates a batch file on the system, then I compiled the script into an executable with PyInstaller. The executable registers itself as a Windows service with automatic startup.

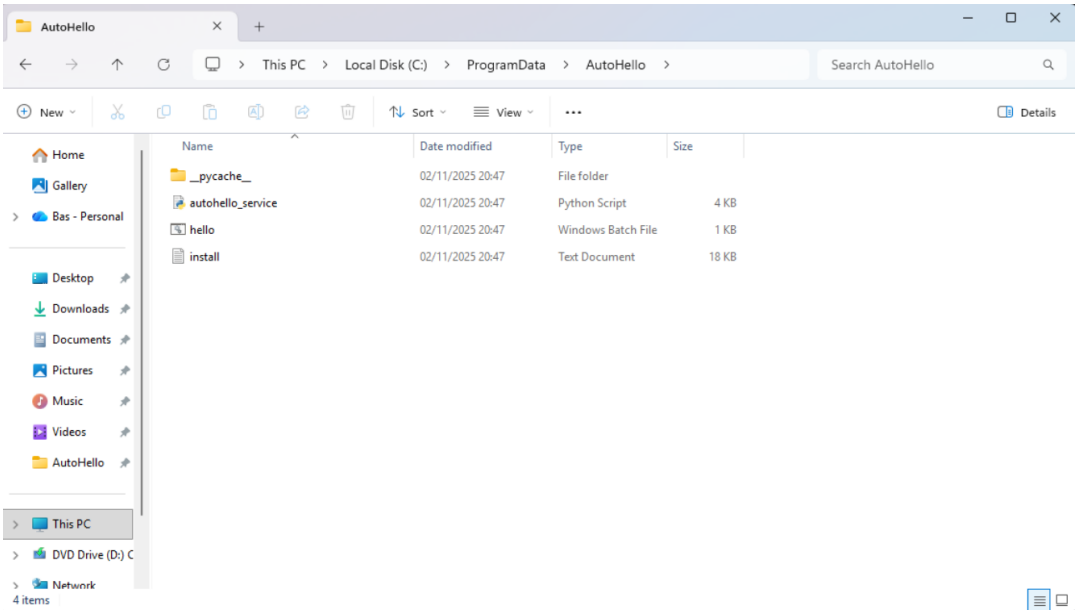


Figure 4.8: Files created by the service

I configured it to start with "Automatic (Delayed Start)" so it runs shortly after login. When I rebooted the system, the service started automatically and executed the batch file, opening a terminal window with a hello message (Figure 4.9). In a real attack, this could run a reverse shell instead of just printing a message.

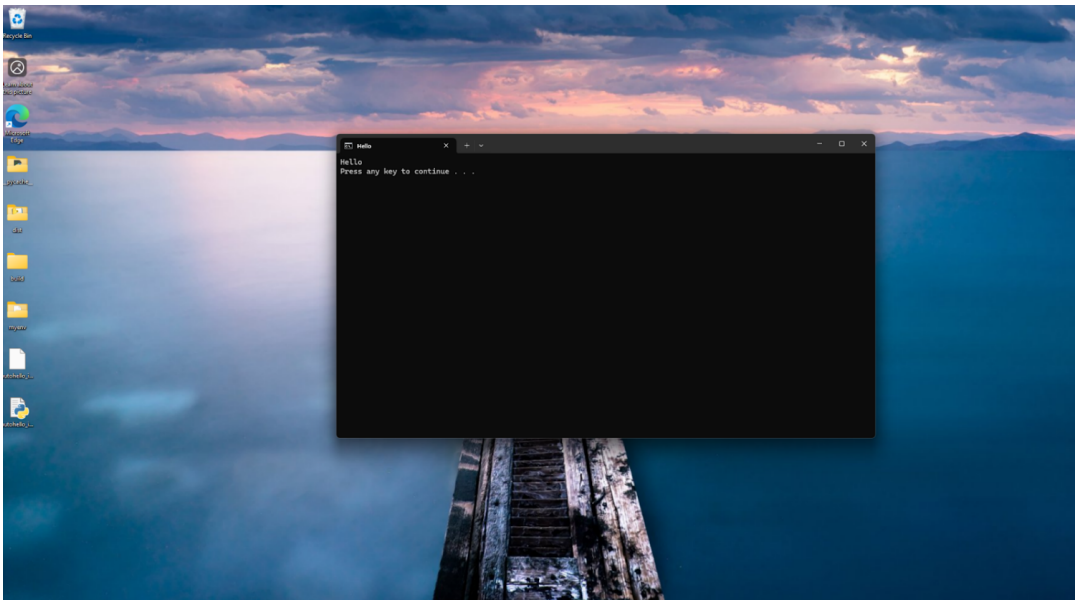


Figure 4.9: Service running at startup

Experiment 5: Registry-Based Persistence

I also tested a better persistence method using the Windows Registry. This approach is actually more practical than the service method because:

- You don't need admin privileges to install it
- It's harder to detect since it blends in with legitimate startup programs
- It works with pure PowerShell, so no Python dependency
- It has automatic reconnection if the connection drops
- It prevents multiple instances from running at once

I tested this with both PowerShell and Python implementations, and both worked perfectly. The PowerShell version is particularly interesting because you can convert the script to a standalone .exe using win-ps2exe, which makes it look like a regular Windows program.

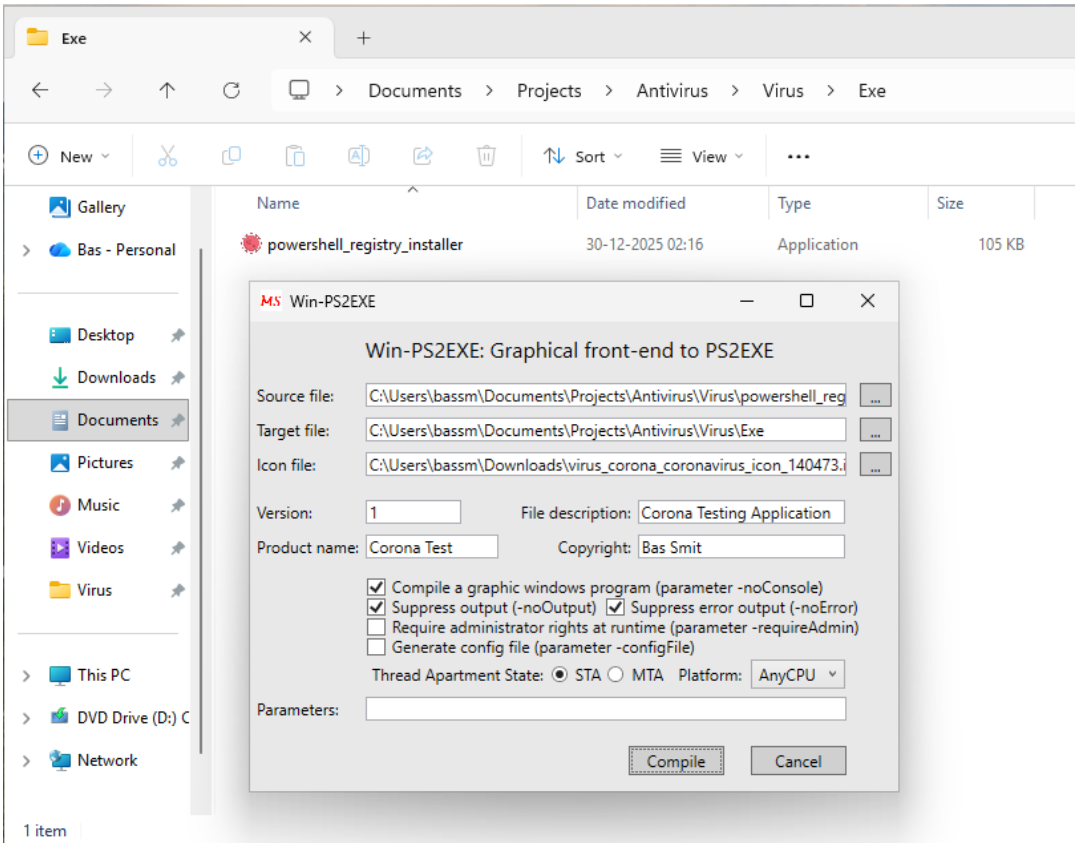


Figure 4.10: Converting the PowerShell script to an executable

The way it works is pretty straightforward. The script creates a hidden folder in AppData and writes the PowerShell payload there. Then it adds an entry to the Registry Run key (`HKCU:\Software\Microsoft\Windows\CurrentV`) with a name like "WindowsUpdateService" that points to the payload script. This makes it execute automatically whenever the user logs in.

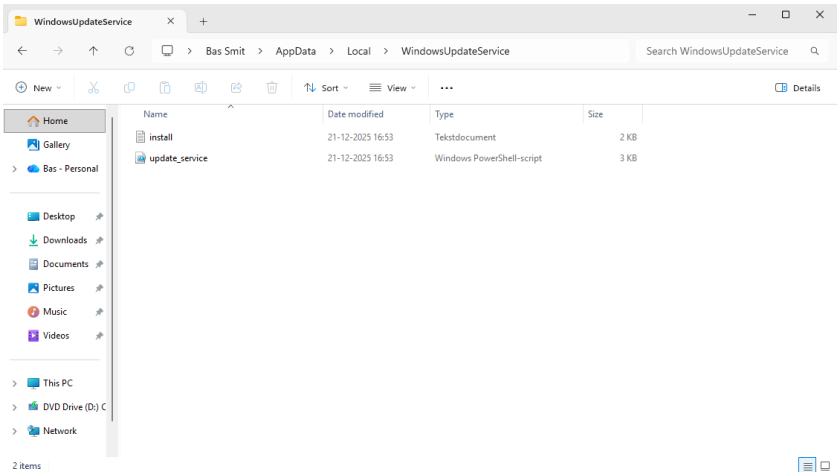


Figure 4.11: Files created in the AppData folder

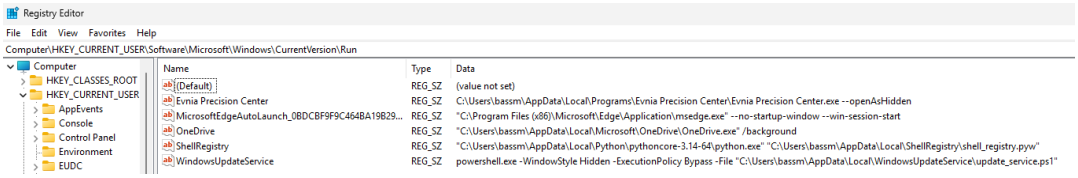


Figure 4.12: Registry entry that makes the payload run at startup

The payload itself is a reverse shell that connects back to my listener. It runs hidden (using `-WindowStyle Hidden`) so the user doesn't see any windows popping up. If the connection drops, it automatically tries to reconnect every 30 seconds. After rebooting and logging back in, the payload executed and I got a shell connection (Figure 4.13).

```
bas@raspberrypi ~ nc -lvnp 5003
listening on [any] 5003 ...
connect to [192.168.178.98] from (UNKNOWN) [192.168.178.192] 11156
dir
 Volume in drive C has no label.
 Volume Serial Number is 3040-19B9

 Directory of C:\

15-09-2025  19:51    <DIR>          inetpub
01-04-2024  07:26    <DIR>          PerfLogs
08-11-2025  15:52    <DIR>          Program Files
15-09-2025  20:02    <DIR>          Program Files (x86)
02-11-2025  13:57    <DIR>          Users
05-12-2025  00:50    <DIR>          Valheim
21-12-2025  15:34    <DIR>          Windows
               0 File(s)              0 bytes
               7 Dir(s)  31.411.961.856 bytes free
```

Figure 4.13: Reverse shell connection through registry persistence

I uploaded the compiled executable to VirusTotal to see how many antivirus engines would detect it. The results were surprising - most major antivirus products including Bitdefender and Malwarebytes didn't flag it as malicious (Figure 4.14). Only some lesser-known engines caught it. This shows that even well-known persistence techniques can evade detection when implemented carefully.

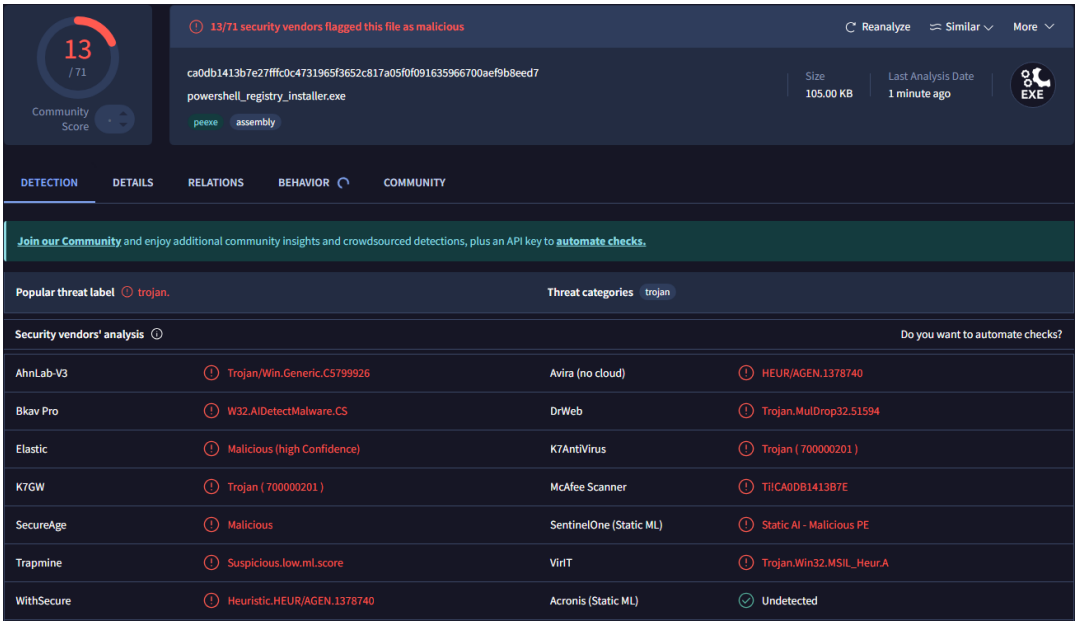


Figure 4.14: VirusTotal results showing low detection rate

Experiment 6: Testing Detection Methods

For the final part, I wanted to compare how different detection methods work. I built a simple antivirus simulator that implements three approaches: signature-based, heuristic-based, and behaviour-based detection.

The signature-based scanner maintains a database of SHA256 hashes from known malware. It scans files in a directory, calculates their hashes, and checks if they match anything in the database. When I ran it on my test files, it correctly flagged the ones with known hashes but completely missed the packed versions since their hashes were different. This really shows the limitation of signature-based detection.

For heuristic detection, I implemented rules that look for suspicious characteristics like packed sections, unusual API calls, or weird metadata. I tested it with files that had suspicious patterns (like registry modification code or unusual system calls) and the scanner generated warnings even though these weren't in the signature database. The downside is it also flagged some legitimate programs that happened to use similar patterns, which demonstrates the false positive problem.

The behaviour-based scanner was the most interesting to implement. It monitors what programs actually do at runtime - things like file encryption, network connections, process injection, or registry modifications. I created test samples that mimicked ransomware behaviour by encrypting multiple files rapidly. The behaviour scanner immediately flagged this as malicious and (in the simulation) terminated the process. This method is really powerful against zero-day threats because it doesn't rely on knowing the malware's signature beforehand.

4.4. Safety and Ethics

Since I was working with malware techniques, I had to be really careful about safety. Everything was done in isolated virtual machines that were completely cut off from my actual computer and the internet. I used NAT networking when I needed to test reverse shells, but the VMs could only talk to each other, not to any external systems.

I didn't use any real malware samples - everything was proof-of-concept code that I wrote myself. The payloads were intentionally simple and only did harmless things like printing messages or opening shells within the lab environment. I made sure nothing could escape the VMs or affect real systems.

All of this research was done for educational purposes to understand how malware works and how to better defend against it. The techniques I demonstrated are already well-documented in the cybersecurity

field. The point was to see how effective they are against different detection methods and to learn how to improve defenses. Obviously using any of these techniques outside a controlled lab environment without authorization would be illegal and unethical.

5

Results

This chapter presents the findings from each experimental procedure, documenting the effectiveness of various evasion techniques and detection methods.

5.1. Steganographic Delivery Results

Steganographic embedding of malicious executables within PDF files remains effective primarily on older or unpatched systems. On fully updated platforms, modern security solutions such as Windows Defender are increasingly capable of detecting and quarantining basic malicious PDF samples due to improved signature coverage and behavioral analysis. However, this reduced effectiveness on up to date systems does not eliminate the overall threat. A significant number of environments continue to operate with outdated operating systems, legacy software, or delayed security updates. In such contexts, malicious PDFs remain highly effective and can easily result in unauthorized code execution and full system compromise when opened by a victim.

5.2. Software Packing Results

Software packing proved to be an effective method for evading basic signature based detection mechanisms. By altering file structure, size, and cryptographic fingerprints, packed executables no longer matched known malware signatures. As a result, static signature scans failed to reliably identify the packed samples.

However, modern antivirus solutions have adapted by incorporating entropy based analysis to detect indicators of compression or encryption commonly associated with packed binaries. Files exhibiting unusually high entropy values may therefore be flagged for further inspection or deeper analysis. While this approach improves detection coverage, it is not a complete solution. Legitimate software such as installers and protectors may also exhibit high entropy, and sophisticated packing techniques can reduce or mask these indicators.

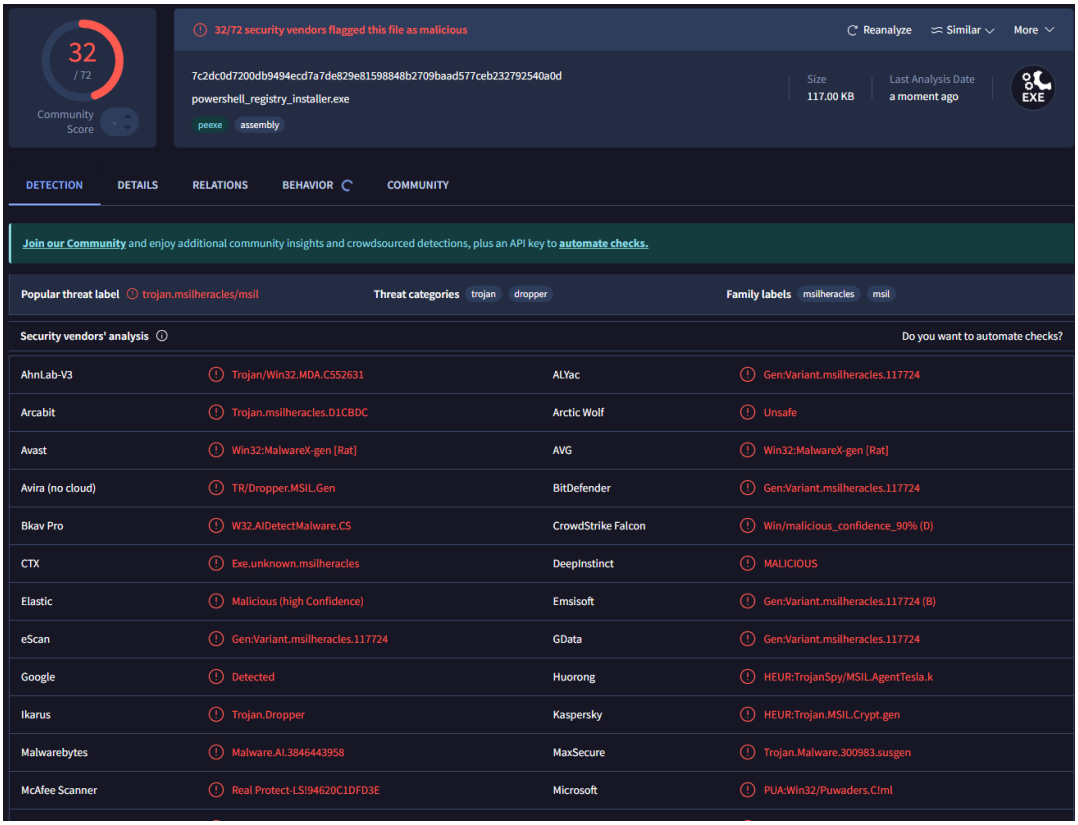


Figure 5.1: VirusTotal scan results showing packed software

Interestingly, in the conducted experiments, the packed executable was detected by a significantly larger number of antivirus engines on VirusTotal than the original, unpacked version. As shown in Figure 4.14, the non packed executable triggered detections from 13 out of 71 engines, whereas the packed variant was flagged by 32 out of 72 engines. This indicates that entropy based heuristics and generic packer detections can increase visibility rather than reduce it, particularly for simple or well known packing techniques. Despite this increased detection rate, software packing remains a relevant evasion method when combined with additional obfuscation layers or more advanced, less recognizable packing implementations.

5.3. Binary Padding Results

Binary padding did not have a significant impact on static file analysis results. Despite substantial increases in file size, static detection engines largely continued to classify the padded executables in the same manner as their unpadded counterparts. Signature based detections and basic heuristic indicators were primarily influenced by the underlying executable content rather than the appended padding data, as the added bytes were located outside the valid executable sections and did not alter program logic.

However, a noticeable effect was observed in the dynamic analysis phase. On VirusTotal, samples with extensive binary padding required more time to be processed by behavior based engines and sandbox environments. The increased file size led to longer upload times, extended preprocessing stages, and delayed execution within analysis sandboxes.

These results indicate that while binary padding alone is ineffective at evading static detection, it can influence the performance and responsiveness of dynamic analysis systems. This suggests that binary padding primarily functions as a resource and timing based evasion technique rather than a direct detection bypass, and its effectiveness is limited unless combined with other obfuscation or evasion methods.

5.4. Persistence Mechanism Results

The persistence mechanisms implemented during this research proved to be highly reliable in maintaining continued execution across system reboots and user logins. Both evaluated approaches, Windows service based persistence and registry Run key based persistence, successfully achieved their intended objective within the controlled laboratory environment.

The Windows service based persistence mechanism demonstrated consistent execution after system restarts. Once installed, the service automatically launched during boot and executed the associated payload without requiring user interaction. From an operational perspective, this method provided strong persistence guarantees. However, the service was clearly visible within the Windows Services management console, making it easier to identify during manual inspection. As a result, while effective, this approach exhibited limited stealth and would likely be detected during routine administrative review or forensic analysis.

The registry based persistence mechanism proved to be both effective and significantly more stealthy. By leveraging the Windows Registry Run key, the payload executed automatically at every user login without requiring administrative privileges. The use of hidden execution parameters and deceptive naming conventions allowed the persistence entry to blend in with legitimate startup items. Across multiple reboot and logout scenarios, the payload consistently re-established execution and successfully initiated outbound connections, confirming reliable persistence.

From a detection perspective, the registry based approach exhibited stronger evasion characteristics. No new services or scheduled tasks were created, reducing the number of observable indicators typically monitored by security tools. Additionally, the PowerShell based implementation avoided dropping large compiled binaries, further limiting signature based detection opportunities. Behavior based detection remained the most effective method for identifying this persistence technique, as repeated registry modifications and automated startup execution represent strong behavioral indicators.

Overall, the results indicate that persistence mechanisms relying on legitimate operating system features remain highly effective when implemented correctly. Service based persistence offers robustness but lower stealth, while registry based persistence provides a more covert and flexible solution. These findings highlight the importance of monitoring startup execution paths and registry modifications, as well as correlating persistence behavior with network activity, to effectively detect and mitigate persistent threats.

5.5. Detection Method Comparison Results

This section presents the experimental results from the three detection methodologies tested: signature-based, heuristic-based, and behaviour-based detection. Each approach was evaluated against the malware samples created in previous experiments.

5.5.1. Signature-Based Detection Results

The signature-based detection tool successfully identified malware samples when their exact hash values were stored in the signature database. Detection was instantaneous and produced no false positives, as the system only flagged files with matching SHA256 hashes.

However, the fundamental weakness of this approach became immediately apparent: any modification to the malware—whether through packing, padding, or even minor byte changes—resulted in a completely different hash value. Once modified, the malware was no longer recognized by the signature database, demonstrating complete evasion. This confirms that signature-based detection, while highly accurate for known threats, offers zero protection against modified or novel malware variants.

The experiment clearly illustrated why signature-based detection alone is insufficient in modern threat environments. Attackers can trivially bypass this method through automated polymorphic techniques, making it essential to layer additional detection mechanisms.

5.5.2. Heuristic-Based Detection Results

Heuristic analysis demonstrated improved detection capabilities compared to pure signature matching. The heuristic module successfully identified suspicious characteristics such as:

- Packed executable sections with high entropy
- Unusual API call sequences indicative of malicious behavior
- Registry modification attempts targeting Run keys
- Suspicious file operations in system directories

Files exhibiting these patterns were flagged even when they did not match any known signatures. This allowed detection of previously unseen malware variants and obfuscated samples.

However, heuristic detection also generated false positives. Legitimate software installers, system utilities, and development tools occasionally triggered alerts due to their use of compression, administrative privileges, or system-level modifications. Balancing detection sensitivity against false positive rates proved challenging, requiring careful tuning of heuristic rules.

The results confirm that heuristic detection provides valuable defense against unknown threats but requires ongoing calibration and cannot operate as a standalone solution.

5.5.3. Behaviour-Based Detection Results

Behaviour-based detection proved to be the most effective methodology across all tested scenarios. The monitoring tool successfully identified malicious activity in real-time, regardless of code obfuscation or structural modifications.

Key detections included:

- Establishment of outbound network connections to suspicious IP addresses
- Rapid modification of multiple files (simulating ransomware encryption)
- Creation of persistence mechanisms through registry modifications
- Execution of hidden PowerShell processes
- Attempts to modify startup configuration

Critically, behaviour-based detection identified both the Windows service and registry persistence mechanisms during execution, despite their use of legitimate operating system features. The ability to correlate multiple indicators—such as registry modification followed by network activity—provided high-confidence detection with minimal false positives.

The primary limitation observed was that detection occurred only after the malware began executing. Unlike signature or heuristic methods that can scan files pre-execution, behaviour-based detection requires runtime analysis. This introduces a window of opportunity for fast-acting malware to cause damage before intervention.

Despite this limitation, behaviour-based detection remains the most robust defense against modern evasion techniques, particularly when combined with automated response capabilities to terminate malicious processes immediately upon detection.

6

Discussion

This chapter interprets the experimental findings in the context of modern cybersecurity practice, examining their implications for both offensive and defensive strategies. The discussion addresses the research questions, evaluates the effectiveness of evasion techniques, and considers practical recommendations for security professionals.

6.1. Effectiveness of Evasion Techniques

The experimental results demonstrate that several evasion techniques remain highly effective against certain security solutions, challenging the assumption that modern antivirus products have universally adapted to common obfuscation methods.

6.1.1. Steganographic Delivery

Steganographic embedding within PDF files proved effective primarily against outdated or unpatched systems. While modern security solutions like Windows Defender increasingly detect basic malicious PDFs, the technique remains viable in environments with delayed updates or legacy software, which is still prevalent in many organizations due to compatibility requirements, operational constraints, or resource limitations. The continued effectiveness in such environments confirms that steganography remains a practical attack vector, particularly when combined with social engineering.

6.1.2. Software Packing Paradox

The packing experiments revealed an unexpected outcome: packed executables triggered more antivirus detections than unpacked versions. This counterintuitive result suggests that modern security solutions have evolved sophisticated entropy-based heuristics specifically designed to identify packed binaries. However, this does not invalidate packing as an evasion technique. It merely indicates that simple, well-known packers like UPX are widely recognized. Custom packers, multi-layer obfuscation, and advanced packing techniques likely remain effective, as evidenced by their continued use in real-world malware campaigns.

6.1.3. Binary Padding Limitations

Binary padding demonstrated minimal impact on static detection but introduced measurable delays in dynamic analysis systems. This suggests padding functions primarily as a resource exhaustion technique rather than a direct detection bypass. Its limited effectiveness as a standalone method confirms that modern antivirus engines focus analysis on executable sections rather than file size, though padding may still contribute to evasion when combined with other techniques or when targeting specific security products with known performance thresholds.

6.1.4. Persistence Mechanisms: A Critical Gap

The persistence experiments revealed the most significant finding of this research: both Windows service-based and registry-based persistence mechanisms successfully evaded Windows Defender in

all tested scenarios. More importantly, the registry-based approach operated without administrative privileges while maintaining reliable execution across reboots and user sessions.

This represents a fundamental security gap. Despite being well-documented attack techniques, persistence mechanisms leveraging legitimate operating system features remain difficult to detect through signature or heuristic methods alone. The registry persistence variant, when converted to an executable using win-ps2exe, evaded detection by major antivirus vendors including Bitdefender and Malwarebytes on VirusTotal, while still maintaining full functionality.

The implication is clear: attackers can achieve persistent access on target systems using straightforward techniques that bypass frontline defenses. This underscores the critical importance of behaviour-based monitoring and endpoint detection and response (EDR) solutions that can identify suspicious persistence establishment regardless of the specific implementation method.

6.2. Detection Method Analysis

6.2.1. Signature-Based Detection: Known Limitations Confirmed

The signature detection experiments confirmed the well-established limitation of hash-based detection: any file modification defeats signature matching. While this is not surprising, it validates the necessity of layered detection strategies. Signature-based detection remains valuable for rapid identification of known threats with zero false positives, but must be supplemented with additional methodologies.

6.2.2. Heuristic Detection: Promising But Imperfect

Heuristic analysis demonstrated the ability to identify suspicious characteristics in unknown samples, successfully flagging packed executables and files with suspicious API usage. However, the challenge of balancing sensitivity against false positives remains unresolved. Legitimate software occasionally exhibits characteristics that trigger heuristic alerts, requiring either reduced sensitivity (decreasing detection rates) or increased analyst workload to investigate false positives.

This trade-off explains why many organizations configure antivirus products conservatively, potentially missing threats to avoid overwhelming security teams with alerts—a situation attackers can exploit.

6.2.3. Behaviour-Based Detection: Essential but Reactive

Behaviour-based detection proved most effective against all evasion techniques, successfully identifying malicious activity regardless of code obfuscation. The ability to detect persistence establishment, network connections, and suspicious process execution provides critical visibility into actual threat behavior.

However, behaviour-based detection operates reactively. Malware must execute before detection occurs. Fast-acting threats like ransomware can cause significant damage in the brief window before detection and response. This limitation necessitates rapid automated response capabilities and proactive security measures (application whitelisting, least-privilege access) to minimize exposure.

6.3. Practical Implications for Organizations

6.3.1. The Inadequacy of Single-Layer Defense

The research demonstrates that no single detection method provides comprehensive protection. Signature-based detection fails against modified malware, heuristic detection generates false positives, and behaviour-based detection operates reactively. Organizations must implement layered defense strategies combining multiple detection methodologies with proactive security controls.

6.3.2. The Persistence Problem

The successful evasion of Windows Defender by persistence mechanisms, particularly registry-based persistence without administrative privileges, represents a significant practical concern. Organizations cannot rely solely on antivirus software to prevent persistent threats. Essential defensive measures include:

- Monitoring registry Run keys and other autostart locations

- Implementing application whitelisting to prevent unauthorized executables
- Deploying EDR solutions with persistence detection capabilities
- Conducting regular security audits of startup configurations
- Restricting user permissions to minimize persistence establishment opportunities

6.3.3. The Role of User Awareness

Steganographic delivery via malicious PDFs relies fundamentally on user interaction. Technical controls alone cannot eliminate this threat—users must be trained to recognize suspicious attachments, verify sender authenticity, and report potential phishing attempts. Security awareness training remains an essential component of defense despite being frequently undervalued.

6.3.4. Resource Constraints and Security Trade-offs

The varying effectiveness of evasion techniques across different environments (updated vs. outdated systems, different antivirus products) highlights the impact of resource constraints. Organizations unable to maintain current patches, upgrade legacy systems, or invest in advanced security solutions face disproportionate risk. Security decision-makers must understand these trade-offs and prioritize investments based on realistic threat assessment rather than assuming all defensive tools provide equal protection.

7

Conclusion

This research investigated modern malware evasion techniques and detection methodologies through practical experimentation in controlled laboratory environments. By implementing and testing steganographic delivery, software packing, binary padding, persistence mechanisms, and three detection approaches, the study provides empirical evidence regarding the effectiveness of both offensive and defensive strategies.

7.1. Key Findings

The experimental results reveal several critical insights that challenge common assumptions about malware detection:

1. Persistence mechanisms remain highly effective. Both Windows service-based and registry-based persistence techniques successfully evaded Windows Defender across all tested scenarios. The registry-based approach, which requires no administrative privileges and can be converted to an executable, also bypassed detection by major antivirus vendors including Bitdefender and Malwarebytes on VirusTotal. This demonstrates a significant gap in current defensive capabilities against techniques that leverage legitimate operating system features.

2. Evasion techniques show varied effectiveness. While some evasion methods (like simple UPX packing) actually increased detection rates due to modern entropy-based heuristics, others (like registry-based persistence and steganographic delivery) remained highly effective. The results indicate that evasion effectiveness depends heavily on implementation sophistication and the specific security controls in place.

3. Detection methodology matters significantly. Signature-based detection proved trivially bypassable through any file modification. Heuristic detection identified some suspicious patterns but struggled with false positive rates. Behaviour-based detection demonstrated the highest effectiveness across all evasion techniques, though it operates reactively after malware execution begins.

4. Legacy systems face disproportionate risk. Steganographic PDF delivery and other evasion techniques remained effective against unpatched or outdated systems. Given that many organizations continue operating legacy software due to compatibility requirements or resource constraints, these environments face substantially elevated threat levels.

5. No single defense suffices. Every detection methodology tested exhibited exploitable weaknesses. Effective defense requires layered strategies combining multiple detection approaches, proactive security controls, and user awareness training.

7.2. Research Question Answers

7.2.1. Main Research Question

How do modern malware techniques evade detection, and what defensive strategies can effectively counter these evasion methods?

Modern malware evades detection by exploiting the inherent limitations of individual detection methodologies and by leveraging legitimate operating system features to blend with normal system behavior. Simple file modifications defeat signature-based detection, while persistence mechanisms using Windows services or registry Run keys bypass heuristic analysis by appearing indistinguishable from legitimate software behavior.

Effective defense requires a fundamentally layered approach. Organizations must implement:

- Multiple detection methodologies (signature, heuristic, and behaviour-based)
- Proactive security controls (application whitelisting, least-privilege access)
- Continuous monitoring of persistence locations (registry keys, services, scheduled tasks)
- Endpoint detection and response (EDR) solutions with behavioral analysis
- Regular security auditing and patch management
- Comprehensive user security awareness training

No single control provides complete protection. Attackers only need to bypass one layer, while defenders must successfully implement and maintain multiple defensive measures.

7.2.2. Sub-question Answers

Q1: How effective is steganography as a malware delivery mechanism?

Steganographic embedding within PDF files remains moderately effective, particularly against outdated or unpatched systems. While modern security solutions increasingly detect basic malicious PDFs through improved signatures and behavioral analysis, the technique continues working in environments with delayed updates—a common reality in many organizations. Combined with social engineering, steganography remains a viable attack vector requiring both technical controls and user awareness training to mitigate.

Q2: Can software packing bypass signature-based antivirus detection?

Software packing successfully defeats exact signature matching, as packing alters file structure and cryptographic hashes. However, modern antivirus products have adapted by implementing entropy-based detection and generic packer signatures. Simple packers like UPX are now widely recognized, actually increasing detection rates in testing. The technique remains relevant for sophisticated attackers using custom or multi-layered packing implementations, but basic packing alone no longer guarantees evasion.

Q3: Does binary padding exploit performance-based scanning limitations?

Binary padding demonstrated limited effectiveness as a standalone evasion technique. Static detection engines analyzed executable content regardless of padding, maintaining similar detection rates. However, padding did introduce measurable delays in dynamic analysis systems, suggesting potential utility as a resource exhaustion technique when combined with other evasion methods. Overall, binary padding alone provides minimal evasion benefit against modern security solutions.

Q4: How easily can malware establish persistence on Windows systems?

Persistence establishment proved remarkably straightforward and alarmingly effective. Both Windows service-based and registry-based persistence mechanisms successfully maintained execution across system reboots while completely evading Windows Defender. The registry-based approach operated without administrative privileges, used hidden execution parameters, and when compiled to an executable, bypassed detection by major antivirus vendors on VirusTotal. This represents a critical security gap, as attackers can achieve reliable persistent access using well-documented techniques that frontline defenses fail to detect.

Q5: What are the relative effectiveness of signature-based, heuristic-based, and behaviour-based detection methods?

- **Signature-based detection**: Highly accurate for exact matches but completely fails against any file modification. Provides zero protection against polymorphic or novel malware. Essential for known threat identification but inadequate as a standalone defense.
- **Heuristic-based detection**: Identifies suspicious characteristics in unknown samples, offering improved coverage against novel threats. However, struggles with false positive rates and can be evaded through careful implementation that mimics legitimate software behavior. Provides moderate additional protection when properly tuned.
- **Behaviour-based detection**: Demonstrated the highest effectiveness across all tested scenarios, successfully identifying malicious activity regardless of code obfuscation or structural modifications. Detected both persistence mechanisms during execution despite their use of legitimate OS features. Primary limitation is reactive nature—detection occurs only after execution begins. Represents the most critical component of modern defense but requires rapid automated response capabilities.

The layered implementation of all three methods, rather than reliance on any single approach, provides optimal protection.

Q6: What defensive strategies should organizations prioritize?

Based on experimental findings, organizations should prioritize:

1. **Behaviour-based monitoring and EDR deployment**: Essential for detecting evasion techniques that bypass signature and heuristic detection. Must include rapid automated response capabilities.
2. **Persistence location monitoring**: Continuous auditing of registry Run keys, Windows services, scheduled tasks, and other autostart locations to detect persistence establishment attempts.
3. **Application whitelisting**: Prevents execution of unauthorized binaries, mitigating risks from both delivered malware and persistence mechanisms.
4. **Least-privilege access controls**: Limiting user permissions reduces opportunities for persistence establishment and constrains malware capabilities.
5. **Regular patching and system updates**: Maintains effectiveness of built-in security features and closes exploitable vulnerabilities.
6. **Security awareness training**: Users represent both the initial attack vector (phishing, malicious attachments) and potential detection layer (reporting suspicious activity).
7. **Layered defense architecture**: Implementing multiple overlapping security controls ensures that single-point failures do not compromise entire security posture.

7.3. Contribution to Cybersecurity Knowledge

This research provides empirical validation of both the continued effectiveness of certain evasion techniques and the critical importance of behavioural detection capabilities. The successful evasion of Windows Defender by straightforward persistence mechanisms, particularly those requiring no administrative privileges, highlights a significant practical vulnerability that security practitioners must address through monitoring and proactive controls rather than relying solely on antivirus software.

The findings challenge the assumption that modern security solutions have universally adapted to common evasion techniques, demonstrating that technique effectiveness varies substantially based on implementation sophistication and specific defensive controls deployed. This reinforces the necessity of continuous security testing and validation rather than assuming defensive adequacy based on vendor claims or general product capabilities.

7.4. Practical Value for Stakeholders

Security analysts and incident responders benefit from understanding which evasion techniques remain effective and why, enabling better threat hunting, indicator development, and investigation

strategies focused on behavioural patterns rather than static signatures.

Security engineers and SOC teams can prioritize deployment of behavioural monitoring, persistence detection, and EDR solutions based on empirical evidence of their superior effectiveness against modern evasion techniques.

Organizations and decision-makers gain realistic assessment of security control effectiveness, supporting informed investment decisions and risk evaluation rather than false confidence in insufficient defenses.

Cybersecurity students and researchers receive practical demonstration of offensive techniques in controlled environments, bridging the gap between theoretical concepts and operational realities while understanding both attacker methodologies and defensive requirements.

7.5. Final Remarks

The cybersecurity landscape remains adversarial and dynamic, with attackers continuously adapting to defensive improvements while defenders work to identify and counter new evasion techniques. This research demonstrates that despite advances in security technology, fundamental gaps persist—particularly regarding detection of malware that leverages legitimate operating system features for persistence and evasion.

The most critical takeaway is that organizations cannot rely on traditional antivirus software alone to protect against modern threats. Persistence mechanisms using well-documented techniques successfully evade major security products, demonstrating that effective defense requires behavioural monitoring, continuous security auditing, proactive controls, and user awareness rather than passive reliance on automated detection tools.

Security is not a product but a process requiring ongoing assessment, adaptation, and layered implementation of multiple defensive measures. Organizations that recognize this reality and invest accordingly will maintain substantially stronger security postures than those relying on single-point security solutions, regardless of vendor reputation or product sophistication.

8

Limitations

While this research provides valuable insights into malware evasion and detection capabilities, several limitations must be acknowledged to properly contextualize the findings and identify areas for future investigation.

8.1. Experimental Scope and Complexity

The experiments employed simplified proof-of-concept implementations rather than sophisticated, production-grade malware. Real-world threats often incorporate multiple evasion techniques simultaneously, employ anti-analysis mechanisms, and utilize custom-developed tools rather than publicly available options like UPX or basic PowerShell scripts. Consequently, the findings may underestimate the capabilities of advanced persistent threat (APT) groups or professional cybercriminal operations that invest substantial resources in evasion research and development.

The research focused on a limited set of evasion techniques—steganography, packing, padding, and persistence mechanisms. Modern malware campaigns frequently employ additional methods such as process injection, rootkit functionality, virtual machine detection, sandbox evasion, time-based detonation delays, and encrypted communication channels. These advanced techniques were not evaluated and may exhibit different detection characteristics than those examined.

8.2. Environmental Constraints

All experiments were conducted within isolated Windows 10 virtual machine environments. This approach ensured safety and ethical compliance but introduced several limitations:

Operating System Diversity: The research did not evaluate malware behavior or detection effectiveness across different Windows versions (7, 8, 11), server operating systems, or alternative platforms (Linux, macOS). Evasion technique effectiveness may vary substantially across different operating systems and their respective security architectures.

Network Isolation: Virtual machines were intentionally isolated from external networks to prevent accidental malware spread. This precluded testing of cloud-based antivirus features, network-based detection systems, threat intelligence integration, and command-and-control communication detection. Many modern security products rely heavily on cloud lookups for threat identification, meaning the detection rates observed may not reflect real-world effectiveness.

Enterprise Environment Representation: Laboratory virtual machines do not replicate the complexity of actual enterprise networks, including domain configurations, group policies, endpoint management systems, network segmentation, intrusion detection systems, and security information and event management (SIEM) platforms. Detection capabilities in managed enterprise environments may differ significantly from isolated test systems.

8.3. Detection Tool Coverage

The research evaluated a limited number of antivirus and security products due to resource and licensing constraints. The findings regarding detection effectiveness should not be generalized to all security solutions, as different vendors implement varying detection algorithms, heuristic rules, and behavioral analysis capabilities. Products not tested may perform substantially better or worse against the evaluated evasion techniques.

Additionally, the study primarily relied on VirusTotal for multi-vendor analysis. While VirusTotal provides broad coverage, it evaluates files in isolation without the behavioral monitoring, network context, and system integration that full product deployments provide. Real-world detection rates may differ from VirusTotal scan results.

8.4. Temporal Constraints

Cybersecurity represents a rapidly evolving field where both attack and defense capabilities advance continuously. This research provides a snapshot of detection effectiveness at a specific point in time (late 2024 / early 2025). Antivirus vendors regularly update signatures, heuristics, and behavioral rules, meaning detection rates for the tested samples may improve over time. Conversely, attackers continuously develop new evasion techniques, potentially rendering current defensive measures less effective.

The persistence mechanisms tested, while currently effective, may be addressed by security vendors through improved behavioral detection or specific countermeasures in future product updates. Findings should be interpreted as representative of current capabilities rather than permanent defensive limitations.

8.5. Ethical and Legal Constraints

Ethical research requirements prohibited testing with real malware samples, deployment of actual command-and-control infrastructure, or evaluation against production systems. While necessary and appropriate, these constraints limited the research's ability to fully replicate real-world attack scenarios and assess defensive effectiveness under operational conditions.

The research also excluded exploitation of genuine vulnerabilities, deployment of malware to test real organizational defenses (even with permission), or participation in adversarial security testing beyond controlled laboratory environments. These limitations ensure ethical compliance but restrict the realism and generalizability of findings.

8.6. Resource Limitations

Financial and time constraints limited the scope of experiments. A more comprehensive study would include:

- Testing against a broader range of commercial security products
- Evaluation of enterprise EDR platforms with behavioral analysis
- Assessment of network-based detection systems
- Analysis of malware behavior across different system configurations
- Testing of more sophisticated evasion technique combinations
- Longitudinal studies examining detection capability evolution over time

8.7. Future Research Directions

Despite these limitations, the research provides valuable practical insights and identifies several important areas for future investigation:

Advanced Evasion Techniques: Future research should evaluate more sophisticated methods including process injection, memory-only execution, fileless malware, and anti-analysis mechanisms to assess detection capabilities against advanced threats.

Multi-Platform Analysis: Expanding research to Linux, macOS, and mobile operating systems would provide broader understanding of evasion technique effectiveness across different security architectures.

Enterprise Environment Testing: Evaluation within realistic corporate networks with full security stack deployment (firewalls, IDS/IPS, EDR, SIEM) would provide more accurate assessment of organizational defensive capabilities.

Real Malware Analysis: Controlled analysis of actual malware samples (rather than proof-of-concepts) in secure research environments would validate findings against sophisticated, professionally developed threats.

Longitudinal Studies: Tracking detection capability evolution over time as vendors update products would provide insights into adaptive security improvement and the ongoing arms race between attackers and defenders.

Detection Bypasses: Investigating methods to evade behavioral detection systems represents a critical research area, as these technologies are increasingly positioned as primary defensive measures.

8.8. Validity Within Constraints

While acknowledging these limitations, the research maintains validity within its defined scope. The findings accurately represent detection capabilities against tested evasion techniques within the specified environment. The practical demonstrations provide educational value for security professionals and students while highlighting genuine defensive gaps that warrant attention.

The successful evasion of Windows Defender by persistence mechanisms, despite environmental limitations, represents a particularly significant finding indicating fundamental detection challenges that likely extend beyond the laboratory environment. Organizations cannot assume that antivirus software alone provides adequate protection against relatively straightforward persistence techniques.

Overall, the limitations identified do not invalidate the research findings but rather contextualize them appropriately and identify valuable directions for continued investigation. The study achieves its intended objectives of demonstrating evasion techniques, evaluating detection methodologies, and providing practical security insights while operating within necessary ethical, legal, and resource constraints.

References

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2012). A survey on automated dynamic malware analysis techniques and tools. *ACM Computing Surveys*, *44*(2), 1–42. https://doi.org/10.1145/2089125.2089126

European Union Agency for Cybersecurity. (2025). *Enisa threat landscape 2025* (Chapter 3, Figure 1). ENISA. Retrieved December 29, 2025, from https://www.enisa.europa.eu/sites/default/files/2025-12/ENISA%20Threat%20Landscape%202025_v1.1.pdf

Sikorski, M., & Honig, A. (2012). *Practical malware analysis: The hands-on guide to dissecting malicious software*. No Starch Press.

Symantec. (2020). *Internet security threat report*. Retrieved December 31, 2025, from https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence

Trend Micro Research. (2023). *Emotet returns, now adopts binary padding for evasion*. Retrieved December 29, 2025, from https://www.trendmicro.com/en_us/research/23/c/emotet-returns-now-adopts-binary-padding-for-evasion.html

Ugarte-Pedrero, X., Santos, I., Bringas, P. G., & Alvarez, G. (2015). Countering entropy-based malware detection. *Computers Security*, *48*, 36–50. https://doi.org/10.1016/j.cose.2014.09.006

You, I., & Yim, K. (2010). Malware obfuscation techniques: A brief survey. *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, 297–300. https://doi.org/10.1109/BWCCA.2010.85