ELEC-H404

Advanced Security Evasion in Windows

Andranik Voskanyan

Cédric Sipakam

Zinar Mutu

Professor

....

Academic Year

2024 - 2025

Faculty

Electrical Engineering

Contents

List of Figures

Introduction

In operating systems, particularly Windows, are equipped with built-in security solutions such as Windows Defender and advanced Endpoint Detection and Response (EDR) tools to protect against malicious activities. However, cyber attackers continuously develop techniques to evade these defenses, using stealthy methods to hide malicious processes, inject code into legitimate applications, and maintain persistence within compromised systems. This project focuses on evaluating the effectiveness of Windows Defender and third-party EDR solutions in detecting and responding to such advanced evasion techniques.

The primary goal of this project is to simulate real-world attack scenarios involving hidden command execution, keyloggers, backdoor deployment, and process injection, while analyzing how well the security tools detect and respond to these threats. By monitoring system behavior, network traffic, and event logs, the project aims to compare the detection capabilities, response times, and overall effectiveness of these security solutions.

Through this work, the project highlights key vulnerabilities in Windows security mechanisms and identifies potential areas for improvement. The outcomes of the experiments not only contribute to understanding modern evasion tactics but also help propose enhanced defensive measures to strengthen Windows systems against future attacks.

Background

Windows Security Landscape

Microsoft Windows is the most widely used desktop operating system in the world, making it a prime target for cyberattacks. To protect users, Windows offers a **layered security architecture**, including:

Layer	**Description**
Windows Defender Antivirus	Signature-based scanning, real-time file monitoring, heuristic behavior analysis
Windows Defender Firewall	Network filtering and application traffic rules
Windows Defender Exploit Guard	Memory protections, attack surface reduction, folder access control
Windows Event Logging	Logs process creation, network events, and privilege escalation
AMSI (Antimalware Scan Interface)	Allows inspection of PowerShell scripts, macros, and dynamic code at runtime
EDR (Endpoint Detection and Response)	Advanced behavioral analysis, telemetry collection, and threat correlation (often third-party tools)

Table 1: Windows Security Layers and Their Functions

However, attackers have evolved equally **advanced evasion techniques** to **bypass or confuse these security layers**. This project focuses on evaluating how well these defenses stand up to modern **stealth techniques** in real-world tests.

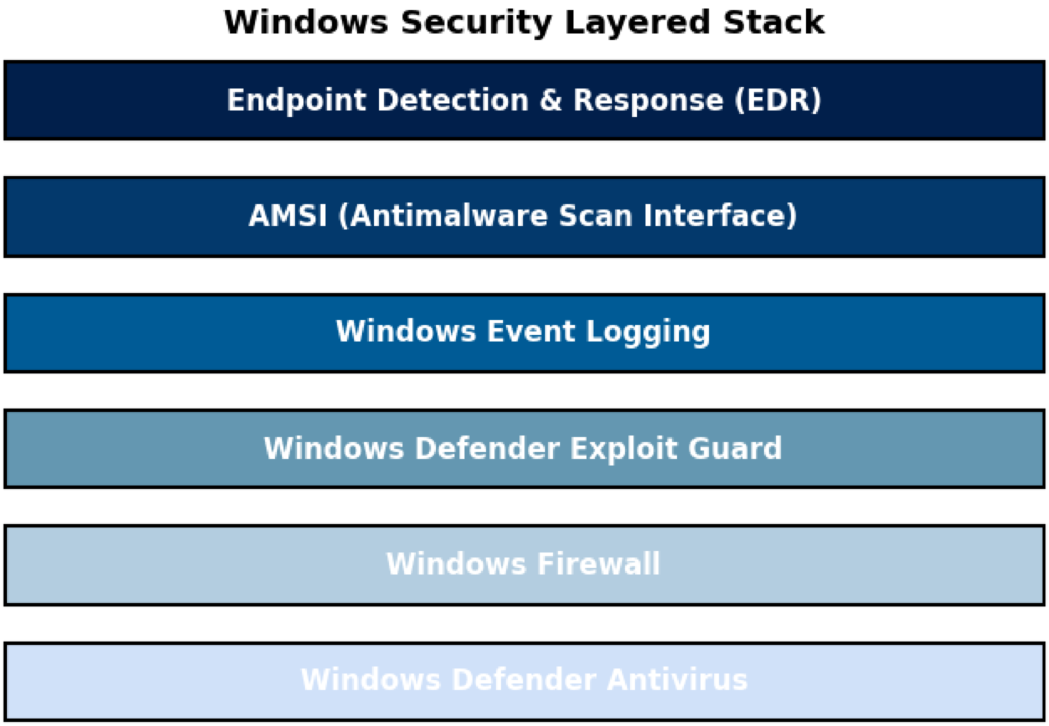
Windows Security Layered Stack
Endpoint Detection & Response (EDR)
AMSI (Antimalware Scan Interface)
Windows Event Logging
Windows Defender Exploit Guard
Windows Firewall
Windows Defender Antivirus

Figure 1: Diagram showing the layered security stack in Windows (Defender, Firewall, AMSI, EDR)

This diagram illustrates the layered architecture of Windows security, starting from basic real-time antivirus scanning up to advanced behavioral analysis provided by EDR solutions. These layers work together to detect known threats, suspicious behaviors, and emerging attacks by combining file scanning, process monitoring, memory inspection, and telemetry analysis.

Evasion Techniques in Windows

Modern attackers rarely use obvious malware files. Instead, they hide within **legitimate processes** or exploit **trusted system tools**. This project focuses on:

1. Hidden Command Execution with LOLBins

Living Off the Land Binaries (LOLBins) are legitimate Windows binaries misused for malicious purposes. Since they are signed by Microsoft, they **bypass many security checks**.

LOLBin	Abuse Example
rundll32.exe	Executes DLL functions directly (payload injection)
mshta.exe	Runs malicious HTML/JS payloads
certutil.exe	Downloads payloads over HTTPS
regsvr32.exe	Loads malicious COM objects remotely

Table 2: Common LOLBins and their abuse techniques

Example command:

```
rundll32.exe shell32.dll,Control_RunDLL payload.dll
```

2. Keyloggers — Silent Data Capture

A **keylogger** hooks into **low-level keyboard events** to capture every keystroke. Simple keyloggers use:

```
SetWindowsHookEx(WH_KEYBOARD_LL, KeyboardProc, NULL, 0);
```

More advanced ones **inject into trusted processes** like ‘explorer.exe‘, making them **blend into normal system behavior**. Detection focuses on:

- Monitoring unexpected process hooks.
- Detecting exfiltration of logged keystrokes.
- Correlating process behavior and file writes.

3. Process Injection & Hollowing

This is one of the most powerful evasion techniques. The attacker injects malicious code into a **trusted process** like `svchost.exe`, effectively hiding the malware.

Injection Type	**Explanation**
DLL Injection	Loads malicious DLL into another process
Process Hollowing	Starts process in suspended state, hollows it, injects malicious code
Thread Hijacking	Injects code into existing process threads

Table 3: Types of Process Injection Techniques

shapes.geometric, arrows

startstop = [rectangle, rounded corners, minimum width=3cm, minimum height=1cm, text centered, draw=black, fill=yellow!30] process = [rectangle, minimum width=3cm, minimum height=1cm, text centered, draw=black, fill=green!15] decision = [diamond, minimum width=3cm, minimum height=1cm, text centered, draw=black, fill=orange!30] arrow = [thick,->,>=stealth]

[node distance=2cm]
(start) [startstop] Start - Attacker Launches Hollowing Process; (createproc) [process, below of=start] CreateProcess() - (Suspended State); (allocmem) [process, below of=createproc] VirtualAllocEx() - Allocate Memory; (writemem) [process, below of=allocmem] WriteProcessMemory() - Inject Payload; (changectx) [process, below of=writemem] SetThreadContext() - Point to Payload; (resumethread) [process, below of=changectx] ResumeThread() - Execute Payload; (end) [startstop, below of=resumethread] Malicious Code Running Inside Trusted Process;
[arrow] (start) – (createproc); [arrow] (createproc) – (allocmem); [arrow] (allocmem) – (writemem); [arrow] (writemem) – (changectx); [arrow] (changectx) – (resumethread); [arrow] (resumethread) – (end);

Process Hollowing Flowchart

This flowchart illustrates the sequence of steps involved in a process hollowing attack. The attacker first launches a process in a suspended state using `CreateProcess()`. Next, memory in the target process is allocated using `VirtualAllocEx()`, and the malicious payload is written into this memory using `WriteProcessMemory()`. The attack then modifies the thread context using `SetThreadContext()` to redirect execution to the malicious payload. Finally, the suspended thread is resumed using `ResumeThread()`, allowing the payload to execute within the context of the trusted process. This technique allows malware to blend into legitimate processes, making detection by traditional security tools more difficult.

4. Backdoor Creation & Persistence Techniques

Attackers want long-term access, so they use **persistence techniques** such as:

Technique	**Example**
Registry Run Key	Adds payload to HKCU\Software\Microsoft\Windows\CurrentVersion\Run
Scheduled Task	Creates hidden task running malware on boot
Service Creation	Registers malware as a system service

Table 4: Persistence Techniques in Windows

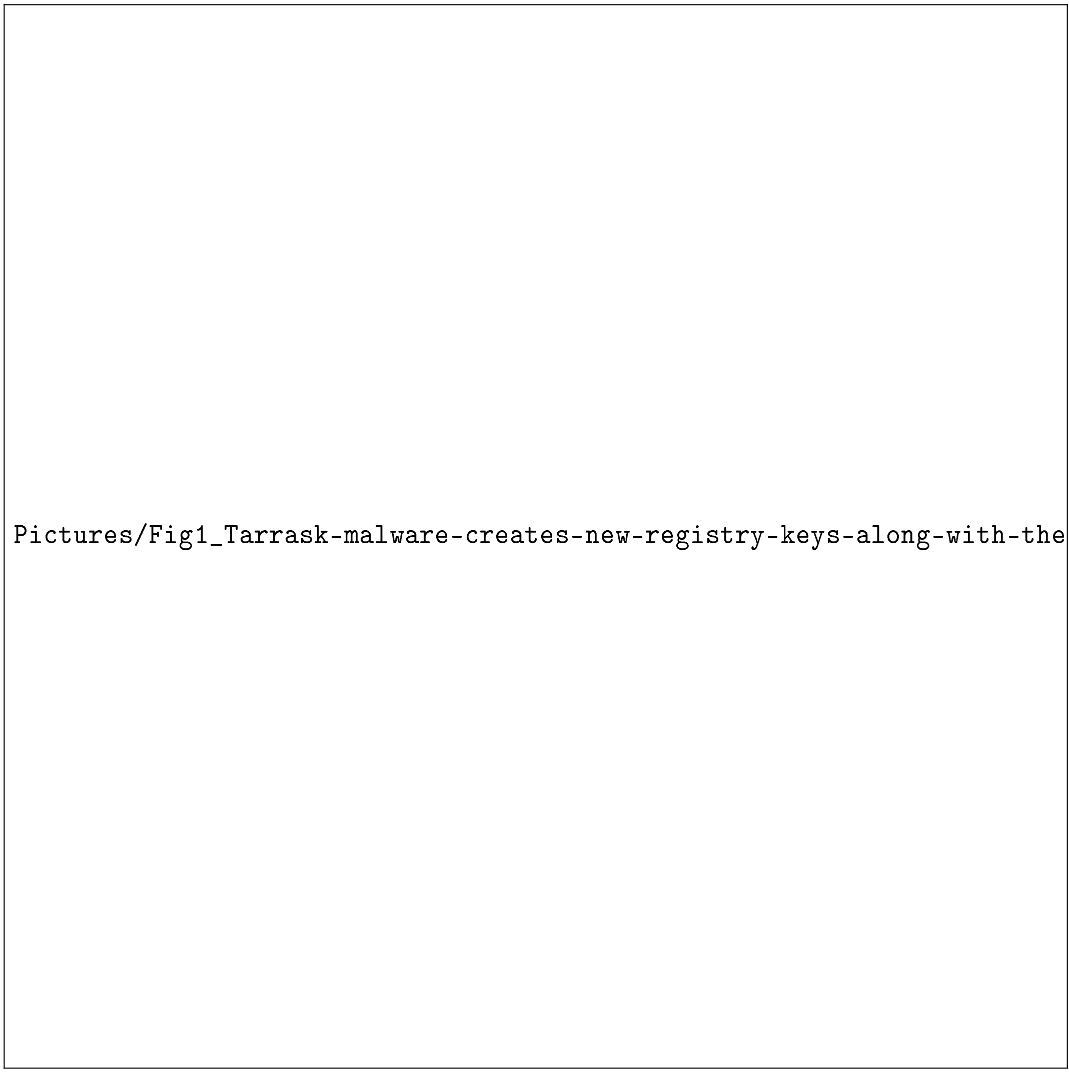
Pictures/Fig1_Tarrask-malware-creates-new-registry-keys-along-with-the-creation-of-new-sc

Figure 2: Screenshot of malicious task created via `schtasks` command in Windows Task Scheduler

Example command:

```
schtasks /create /tn "Updater" /tr "C:\backdoor\payload.exe" /sc onlogon /ru SYSTEM
```

Defender and EDR — Strengths & Weaknesses

1. Detection Methods Used

- Signature Matching: Known bad files, hashes
- Heuristic Analysis: Process behavior (e.g., spawning cmd.exe from Word)
- Log Analysis: Event logs (e.g., suspicious command lines)
- Memory Scanning: Scans process memory for injected code
- Network Monitoring: Detects unusual outbound connections

2. Key Limitations

Limitation	Why it Matters
Trusted Process Blind Spot	Signed Windows processes are trusted by default
Fileless Attack Resistance	If malware never touches disk, signature scans miss it
Log Overload	In noisy environments, alerts get buried
Cross-Process Visibility	Process injection can bypass monitoring hooks

Table 5: Weaknesses in Defender and EDR

Why Real-World Testing Matters

Lab tests often rely on known malware samples, but **real attackers hide in plain sight**:

- PowerShell obfuscation
- Fileless payload delivery
- Encrypted Command & Control (C2)

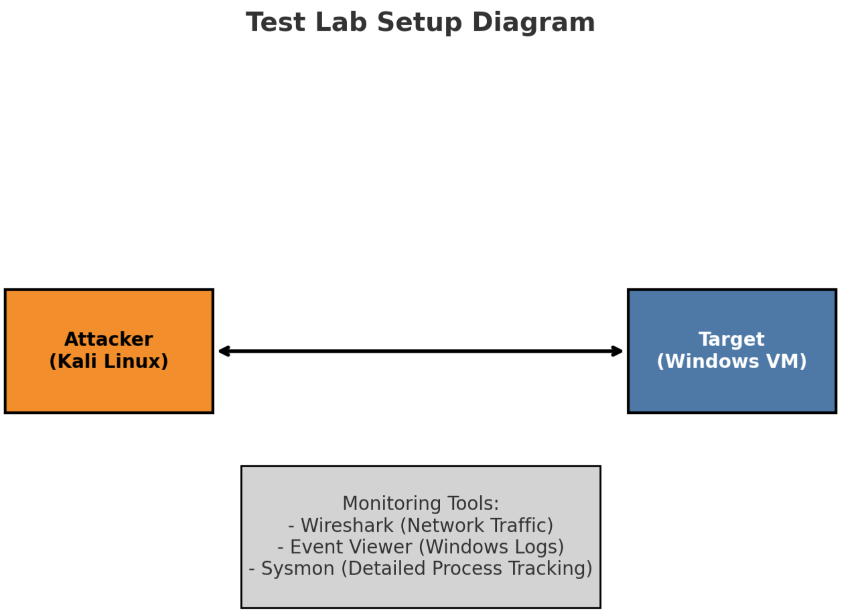
Test Lab Setup Diagram
Attacker (Kali Linux)
Target (Windows VM)
Monitoring Tools:
- Wireshark (Network Traffic)
- Event Viewer (Windows Logs)
- Sysmon (Detailed Process Tracking)

Figure 3: Test Lab Setup: Attacker (Kali Linux) attacking Target (Windows) with monitoring via Wireshark, Event Viewer, and Sysmon

This project simulates these real-world **stealth techniques** to measure:

Evaluation Metric	Explanation
Detection Time	How fast is the initial detection?
Detection Accuracy	How many techniques were detected?
Alert Quality	Are alerts actionable, or just noise?

Table 6: Evaluation Metrics for Real-World Testing

Description

Experimental Results

Introduction for Keylogger

Keyloggers are among the most commonly used tools in offensive security, whether for unethical spying or legitimate penetration testing. This experiment focuses on building and testing a stealth keylogger using Python on Windows, emphasizing its capabilities to remain undetected by built-in security systems and exfiltrate collected data to a remote server. The goal was to demonstrate how attacker tools can be crafted with minimal code and maximum stealth, giving insight into the offensive side of Operating Systems Security.

Keylogger as a Stealth Attack Tool

The keylogger discussed in this report was developed using Python and compiled into an executable with PyInstaller. It operates entirely in user-mode and employs several stealth techniques to avoid detection:

- Logs keystrokes and active window titles
- Captures clipboard content
- Takes periodic screenshots
- Runs in the background with no visible window
- Adds itself to the Windows Startup folder for persistence
- Sends collected logs to a Kali Linux machine over a TCP connection

Bypassing Detection Mechanisms

Despite being simple in design, the keylogger was not flagged by Windows Defender in multiple test environments. This highlights the weaknesses of signature-based detection tools. The use of legitimate Python libraries (e.g., `pynput`, `pyperclip`, `Pillow`), absence of suspicious behavior, and use of ‘–noconsole‘ flag make it appear harmless to default security solutions.

Persistence and Startup

The executable copies itself to the user’s Startup folder under the name `winupdater.exe`. This ensures it runs automatically on every login, without needing admin rights. This simple persistence method is widely used in real-world malware.

How to Use the Keylogger

Installation and Setup

1. Open a terminal on Windows and install Python packages:

```
pip install pynput pywin32 pyperclip Pillow

# If pip is not recognized:
python -m pip install pynput pywin32 pyperclip Pillow
```

2. Save the full keylogger script as `keylogger.py` using any text editor (e.g., Notepad).

3. Open a terminal (Command Prompt or PowerShell) in the folder where `keylogger.py` is located and run:

   ```
   python -m PyInstaller --onefile --noconsole keylogger.py
   ```

   This will create a `dist` folder containing the file `keylogger.exe`.

4. To run the keylogger, open Command Prompt, navigate to the `dist` folder, and run:

   ```
   cd dist
   keylogger.exe
   ```

5. The script will automatically:

   - Start logging keystrokes into `logs/keylogs.txt`
   - Save clipboard content
   - Take screenshots every 60 seconds
   - Copy itself to the Startup folder as `winupdater.exe`
   - Open a TCP socket to send logs to a listening Kali machine

6. **Ensure TCP exfiltration works:** The Windows system must be able to reach the Kali machine's IP. Use this command in CMD to test:

   ```
   ping <kali-ip-address>
   ```

   Replace `<kali-ip-address>` with your actual Kali IP (e.g., 192.168.55.X).

7. On the Kali machine, open a terminal and run the TCP listener before the keylogger exits:

   ```
   nc -lvnp 4444 > received_logs.txt
   ```

   This will listen for the incoming TCP connection and save the received log data into `receivedlogs.txt`.

8. To stop the keylogger, press the `q` key twice in any input field. The logs will then be sent to Kali over the TCP connection.

Keylogger Features

- Logs keystrokes to `logs/keylogs.txt`
- Captures clipboard content
- Takes screenshots every 60 seconds
- Stealth mode via `-noconsole`
- Self-replication to Startup folder for persistence
- TCP exfiltration of logs to remote Kali Linux machine
- Kill-switch with double `q` press

Test Results

- Windows Defender did not detect the executable during or after execution
- TCP exfiltration succeeded from Windows to Kali using `nc`
- Screenshots and clipboard content were accurately captured
- Log file was successfully transferred upon exit

Ethical and Security Implications

This practical demonstration reveals how even non-complicated scripts can act as effective attack tools when they exploit weak points in system behavior and antivirus limitations. While this project serves an educational purpose, it also underscores the need for strong behavioral detection, user awareness, and advanced endpoint monitoring.

Conclusion

This experiment highlights the feasibility of implementing a stealthy, persistent, and network-aware keylogger using basic tools. Understanding the construction and behavior of such malware allows defenders to better anticipate, detect, and neutralize threats in real-world systems.

Future Work

Future work could involve:

- Conducting a comparative analysis of detection rates across different antivirus solutions
- Conducting a user study to assess the effectiveness of user awareness training in preventing keylogger attacks
- Collaborating with cybersecurity professionals to develop best practices for detecting and mitigating keylogger threats

- Exploring the legal and ethical implications of keylogger development and usage

Discussion and Conclusion
References

1. **Lolbins:** https://socprime.com/blog/what-are-lolbins/
2. **Scheduled Task:** https://www.microsoft.com/en-us/security/blog/2022/04/12/tarrask-malware-uses-scheduled-tasks-for-defense-evasion/