

**Operating Systems and Security**

---

# Advanced Security Evasion in Windows with Hidden Commands

---

*Authors :*

Zinar MUTLU

Andranik VOSKANYAN

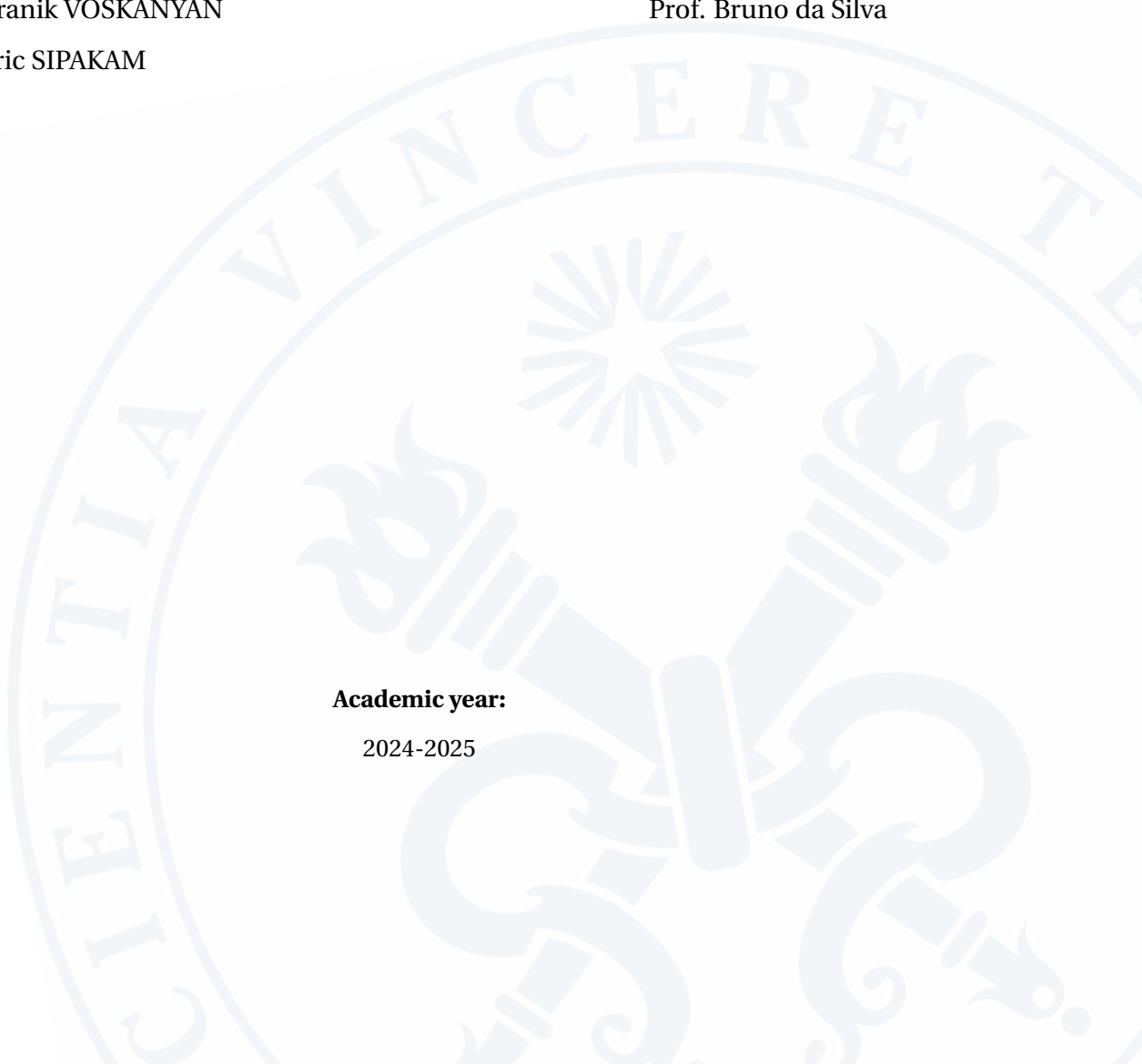
Cedric SIPAKAM

*Professor :*

Prof. Bruno da Silva

**Academic year:**

2024-2025



## Contents

<b>Introduction</b>	<b>1</b>
<b>Background</b>	<b>2</b>
Windows Security Landscape . . . . .	2
Evasion Techniques in Windows . . . . .	3
1. Hidden Command Execution with LOLBins . . . . .	3
2. Keyloggers — Silent Data Capture . . . . .	4
3. Process Injection & Hollowing . . . . .	4
4. Backdoor Creation & Persistence Techniques . . . . .	5
Defender and EDR — Strengths & Weaknesses . . . . .	6
1. Detection Methods Used . . . . .	6
2. Key Limitations . . . . .	7
Why Real-World Testing Matters . . . . .	7
<b>Description</b>	<b>9</b>
<b>Experimental Results</b>	<b>10</b>
Introduction for Keylogger - ** REVIEW - TO BE REWRITTEN ** . . . . .	10
Keylogger as a Stealth Attack Tool . . . . .	10
Bypassing Detection Mechanisms . . . . .	10
Persistence and Startup . . . . .	10
How to Use the Keylogger . . . . .	11
Installation and Setup . . . . .	11
Keylogger Features . . . . .	11
Test Results . . . . .	11
Ethical and Security Implications . . . . .	12
Conclusion . . . . .	12

<b>Discussion and Conclusion</b>	<b>13</b>
<b>References</b>	<b>14</b>

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.

## 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.

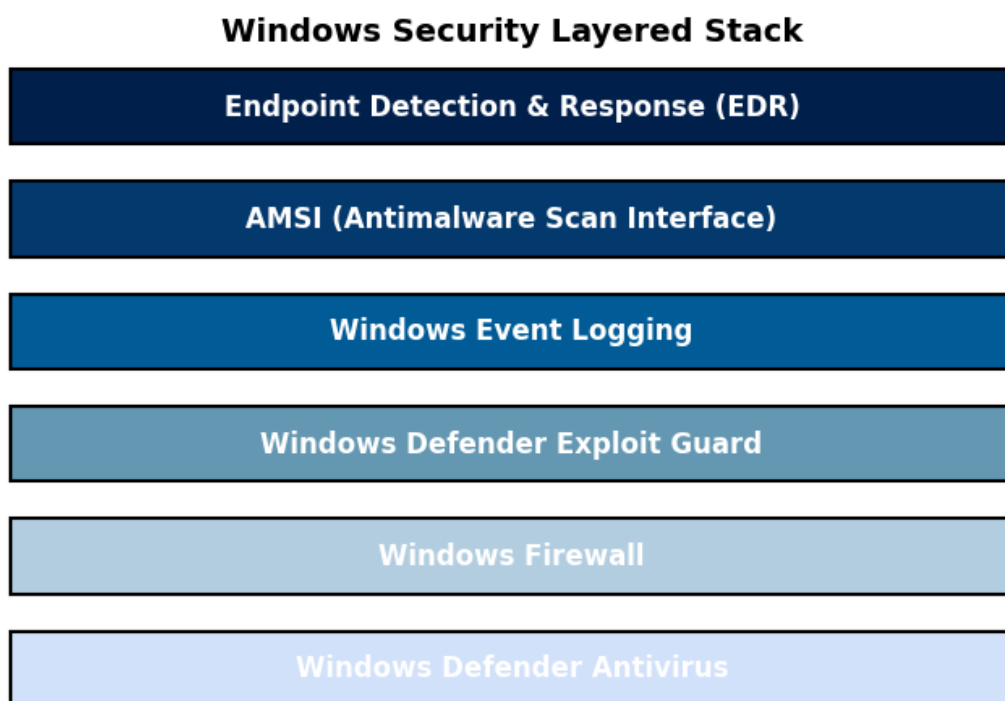


Figure 1: Diagram showing the layered security stack in Windows (Defender, Firewall, AMSI, EDR)

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:

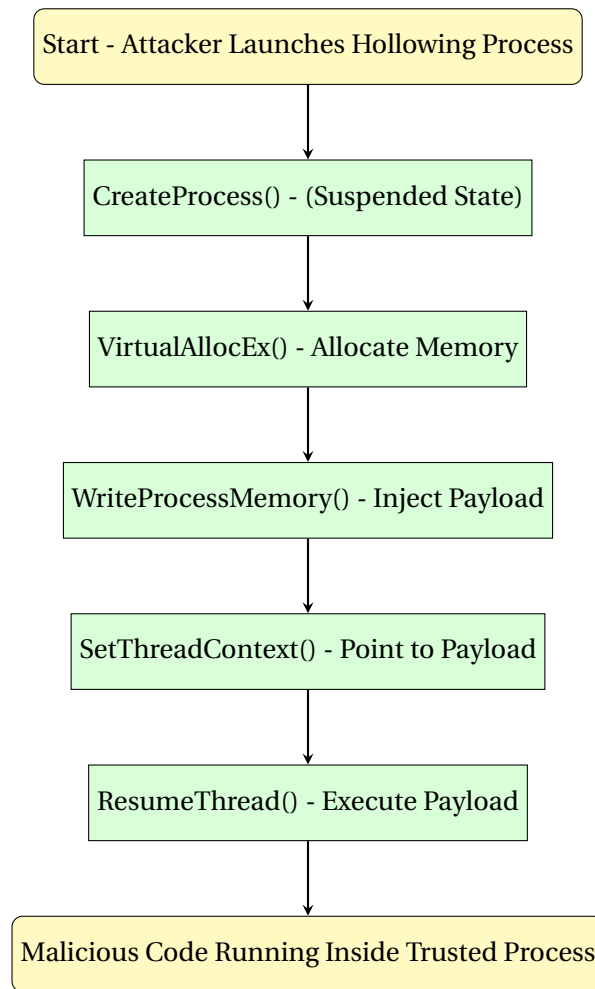
- Monitoring unexpected process hooks.
- Detecting exfiltration of logged keystrokes.
- Correlating process behavior and file writes.

## 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



**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



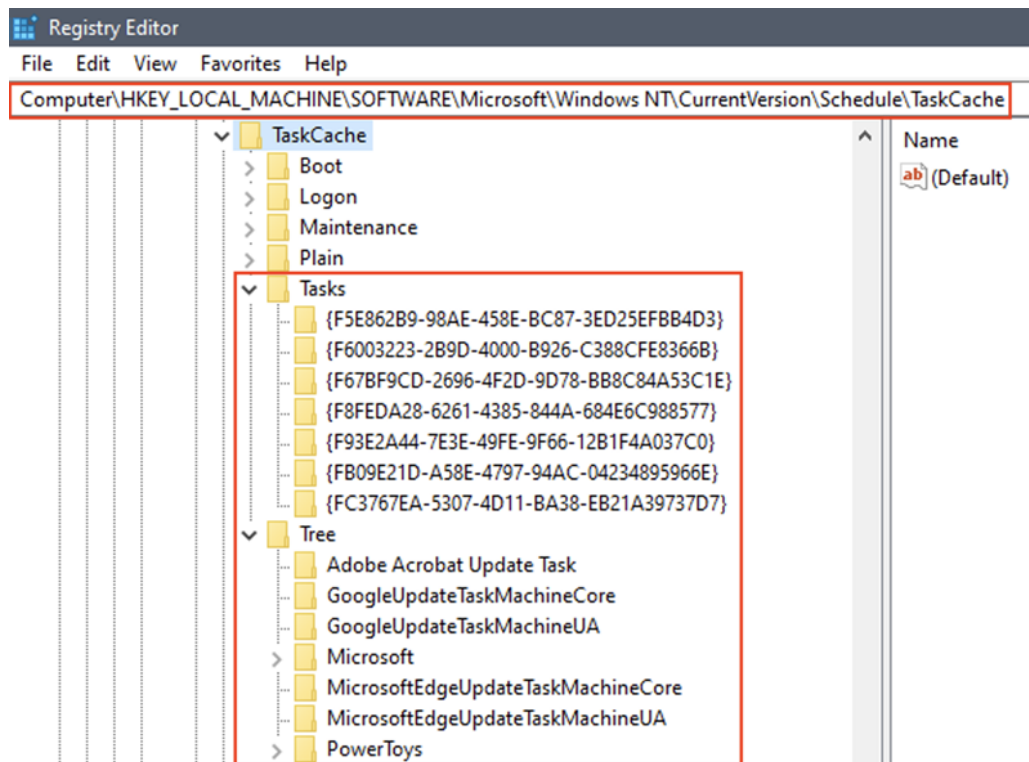


Figure 2: Screenshot of malicious task created via schtasks command in Windows Task Scheduler

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

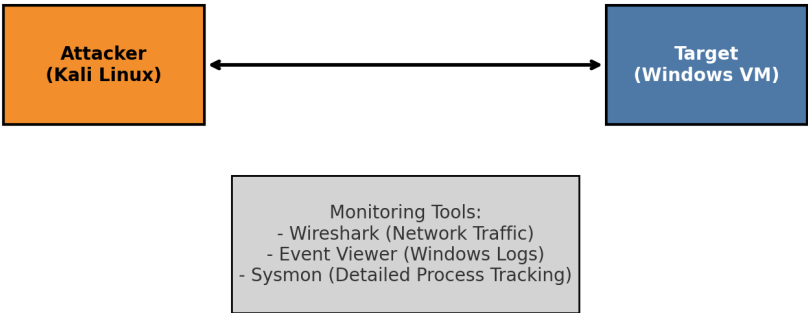


Figure 3: Test Lab Setup: Attacker (Kali Linux) attacking Target (Windows) with monitoring via Wireshark, Event Viewer, and Sysmon

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

<b>Evaluation Metric</b>	<b>Explanation</b>
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
-------------

### **Introduction for Keylogger - \*\* REVIEW - TO BE REWRITTEN \*\***

Keyloggers are one of the most common forms of malware used in both criminal cyberattacks and red-team cybersecurity testing. They log keystrokes entered by a user, often capturing sensitive information such as passwords, emails, chats, and even cryptographic keys. This report explores a practical implementation of a stealthy keylogger on Windows, with an emphasis on bypassing detection mechanisms, achieving persistence, and evaluating its implications for 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

### **Bypassing Detection Mechanisms**

Despite being simple in design, the keylogger was not flagged by Windows Defender in multiple test environments. This reveals a significant gap in signature-based AV systems. Features like the use of legitimate Python libraries (e.g., `pynput`, `pyperclip`, `PIL`), absence of obvious payloads, and stealth execution via `-noconsole` compilation help it evade detection.

### **Persistence and Startup**

The executable self-copies to the user's Startup folder with a disguised name (e.g., `winupdater.exe`). This allows it to run at every login without requiring administrative privileges. Alternative persistence methods like registry modification or scheduled tasks could further enhance its stealth.

# How to Use the Keylogger

## Installation and Setup

1. Install Python 3 and the required libraries by running:

```
pip install pynput pywin32 pyperclip Pillow
```

2. Save the Python script as `keylogge.py` in your preferred directory.
3. Compile the script into a hidden executable using PyInstaller:

```
pyinstaller --onefile --noconsole keylogger_advanced.py
```

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

4. Move the `keylogger.exe` file to a location of your choice.
5. To run the keylogger, double-click the `keylogger.exe` file. It will start logging keystrokes and taking screenshots.

## Keylogger Features

- Start logging keystrokes into `logs/keylogs.txt`
- Capture screenshots every 60 seconds
- Save clipboard contents
- Automatically copy itself to the Windows Startup folder with the name `winupdater.exe`
- To stop the keylogger manually, press the `k` key twice in any input field

## Test Results

- Windows Defender did not detect or quarantine the executable
- Logs and screenshots were captured without interruption
- The Startup entry was successfully created and loaded upon reboot

## **Ethical and Security Implications**

This demonstration underlines how even basic scripting can be weaponized when combined with knowledge of OS internals. The same techniques can be used for both malicious activity and legitimate penetration testing. Therefore, understanding and defending against such tools is critical in the context of Operating Systems Security.

## **Conclusion**

This chapter demonstrates how a stealth keylogger can be used as a reliable, persistent, and undetectable tool under standard configurations. It emphasizes the need for behavioral detection, endpoint monitoring, and proper user education to detect and mitigate such threats.

## 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/>