Exhibit B: Forensic and Technical Findings

Evidence from VHDX of Infected Thumbdrive

The Plaintiff created a virtual disk image (VHDX) of an ostensibly empty thumbdrive that was connected to an infected device. Analysis of the resulting 2 MB image revealed the following:

- Executable Signatures:
    - The image contains multiple MZ headers, the magic number identifying MS-DOS/Windows PE executables (EXE/DLL/EFI).
    - Presence of PE structures later in the dump confirms that at least one embedded executable payload exists within raw sectors.
    - Three distinct MS-DOS executables were identified in the image, despite the thumbdrive appearing empty.
- Reliability Caveat:
    - The Plaintiff notes that the image was created on a machine itself suspected of compromise. Accordingly, while the findings are consistent with malicious payloads, absolute reliability cannot be guaranteed; preliminary findings indicate a high probability of compromise, necessitating professional discovery.
- Indicators of Malicious Characteristics:
    - Compiler/Packer signature matches (via DIE analysis).
    - High entropy values, suggesting encrypted or compressed content.
    - Evidence of obfuscation techniques designed to conceal file contents.
    - Detection of XOR operations within the executable code, a common method of hiding or encoding malicious instructions.

Evidentiary Note

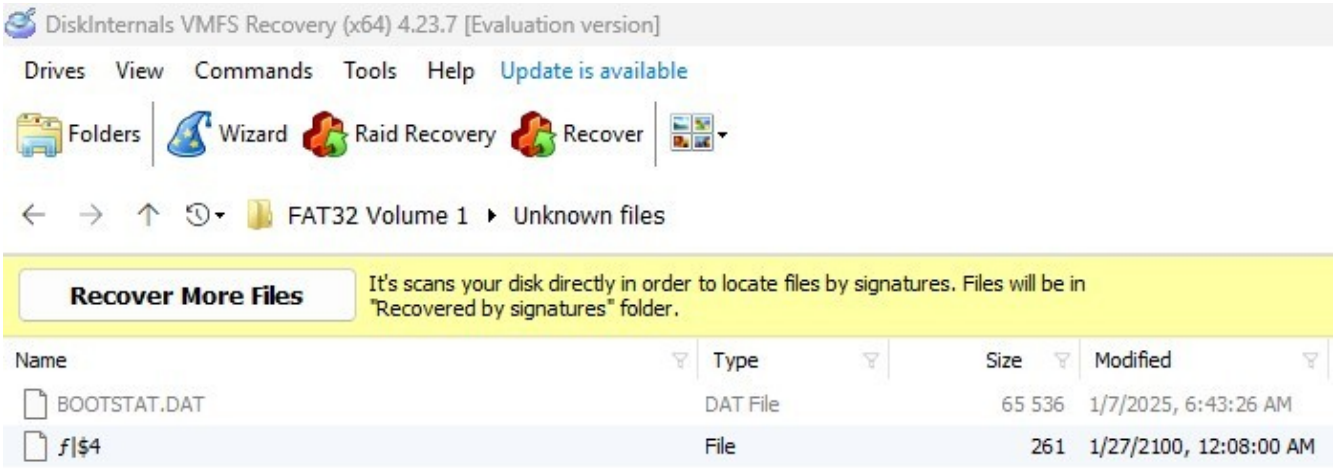
These findings demonstrate that the “empty” thumbdrive contained hidden executable payloads embedded directly in raw disk sectors, consistent with malware infection and obfuscation. The anomalous presence of three executables in a 2 MB image supports the Plaintiff’s claim of repeated intrusion and compromised digital environment.

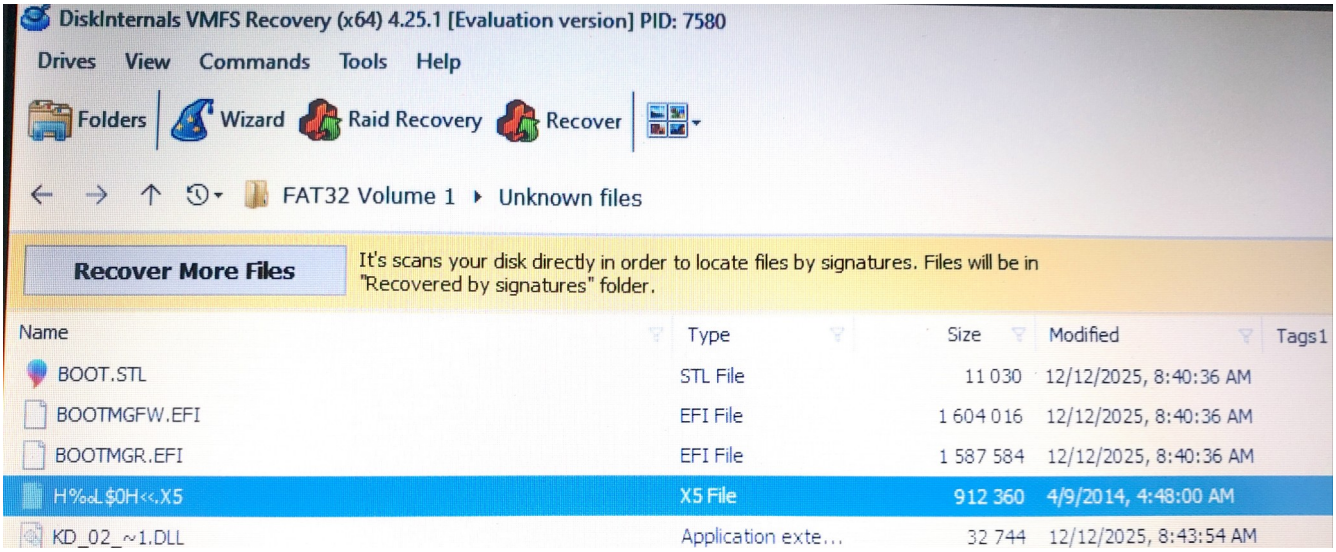
Additional Notes

- Suspected fast-flux C2 infrastructure, and local subnet obfuscation (.arpa routing).
- UEFI rootkits (e.g., LoJax, MoonBounce, CosmicStrand) are stealthy, survive OS reinstalls, and evade AV/EDR.
- NIST SP 800-193 mandates detection, protection, and recovery for compromised firmware.
- NSA/CISA advisory (AA25-093A) warns about fast-flux infrastructure used for resilient C2.

Malware Evidence within EFI Partition

DiskInternals VMFS Recovery was used to detect hidden malware payloads stored within the EFI system partition on infected devices using FAT file signature recovery. These filenames usually contain abnormal characters and carry peculiar modified dates.





These files are absent prior to infection, and then appear after Wi-Fi is enabled in proximity of an infected network.

Malware Evidence Recovered from Numerous Hidden Data Partitions

Some hidden files recovered using file signature recovery via Puran File Recovery seem malicious and are unrecognized as belonging to the Plaintiff. For instance, under Vol9, GRUB bootloader files are listed – this is from a Windows machine *without Linux installed*, indicating a hijacked boot process. There are other seemingly malicious files, such as “ldierWea.pon”, “trolgro.up”, and “torIS5_E_1.EEE” (a file which might indicate that the malware communicates via the Tor network). The following is a limited selection of the files recovered.

```
|____|____|____|____|____recovered
|____|____|____|____|____|____krnln.fnr
|____|____|____|____|____|____PhyDMACC.dll
|____|____|____|____|____|____Vol3
|____|____|____|____|____|____|____ueskeyva.lue
|____|____|____|____|____|____|____Univers_1
|____|____|____|____|____|____|____key.Er_1.ror
|____|____|____|____|____|____|____e-parame.ter
|____|____|____|____|____|____|____setPosit.ion
|____|____|____|____|____|____|____PVSCSI_3.SYS
|____|____|____|____|____|____|____dDeviati.onY
|____|____|____|____|____|____|____Promethe.ans
|____|____|____|____|____|____|____imedoutto
|____|____|____|____|____|____Vol4
|____|____|____|____|____|____|_____site_vi.sit
|____|____|____|____|____|____Vol9
|____|____|____|____|____|____|____GRUBX64.EFI
|____|____|____|____|____|____|____GRUB.CFG
|____|____|____|____|____|____Vol7
|____|____|____|____|____|____|____geWithDe.lay
|____|____|____|____|____|____|____Browser..cpp
|____|____|____|____|____|____Vol1
|____|____|____|____|____|____|____TXTSETUP.OEM
|____|____|____|____|____|____|____refusedto
|____|____|____|____|____Vol1 copy
|____|____|____|____|____|____Vol5
|____|____|____|____|____|____|____BADINTE.GER
|____|____|____|____|____|____|____PFrameWn.dEx
|____|____|____|____|____|____|____teGuidEr.ror
|____|____|____|____|____|____Vol13
|____|____|____|____|____|____|____biodigit.al
|____|____|____|____|____|____|____FOXCONN.PY
|____|____|____|____|____|____Vol25
|____|____|____|____|____|____|_____loop_co.unt
|____|____|____|____|____|____Vol23
|____|____|____|____|____|____|____ldierWea.pon
|____|____|____|____|____|____Vol12
|____|____|____|____|____|____|____300seco.nds
|____|____|____|____|____|____|____nfoObser.ver
|____|____|____|____|____|____|____trolgro.up
|____|____|____|____|____|____Vol1
|____|____|____|____|____|____|____namespa_1.ce
|____|____|____|____|____|____|____VMSCSI.CAT
|____|____|____|____|____|____Vol6
|____|____|____|____|____|____|____PRIMARY.KEY
|____|____|____|____|____|____Vol8
|____|____|____|____|____|____|____nagerCli.ent
|____|____|____|____|____|____|____TimeChoo.ser
|____|____|____|____|____|____|____riteRequ.est
```

```
|____|____|____|____|____|____|____derV2Sha.dow
|____|____|____|____|____|____|_____win_dw..cpp
|____|____|____|____|____|____HDD
|____|____|____|____|____|____|____Vol4
|____|____|____|____|____|____|____|____windows1.256
|____|____|____|____|____|____|____|____yEngine..dll
|____|____|____|____|____|____Vol17
|____|____|____|____|____|____|____torIS5_E_1.EEE
|____|____|____|____|____|____|____kWakeL.ock
|____|____|____|____|____|____|____Restorat.ion
|____|____|____|____|____|____|____satisfi.ed
|____|____|____|____|____|____|____TimePerc.ent
|____|____|____|____|____|____|____RSASHA.256
|____|____|____|____|____|____|____portRequ.est
|____|____|____|____|____|____Vol16
|____|____|____|____|____|____|____modversi.ons
|____|____|____|____|____|____|____KD65AE~1.DL
```

Suspicious Items Summary (Forensic Observations)

The hidden volumes detected via Puran File Recovery contain numerous anomalous and suspicious files, including executables with misleading extensions, UPX-compressed DLLs, multiple non-standard EFI bootloaders, MATLAB files mislabeled as Windows INF drivers, Python scripts in boot partitions, and a large number of algorithmically generated or truncated filenames. These patterns are inconsistent with a normal Windows installation and warrant further forensic examination.

Executables with misleading or nonstandard extensions

- krnln.fnr – PE32 DLL
- PhyDMACC.dll – PE32 DLL, UPX compressed
- AGPSupp.ort – DOS COM executable
- atefail.ed – DOS COM executable

Multiple EFI/bootloader binaries across many volumes

- BOOTX64.EFI (repeated across Vol1, Vol3, Vol4, Vol5, Vol6, Vol7, Vol8, Vol9)
- BOOTX64_1.EFI, BOOTX64_2.EFI, BOOTX64_3.EFI
- BOOTIA32.EFI
- GRUBX64.EFI, GRUBIA32.EFI
- UNICODE.PF2 and UNICODE_1.PF2 reported as PE32+ executables
- SECURE~1.EFI

Secure Boot Bypass: Legitimate EFI (Extensible Firmware Interface) files are the first code to run during system startup. An unsigned EFI file in a boot-related path is a primary indicator of a Bootkit, designed to subvert the Windows Secure Boot process and gain kernel-level persistence before the antivirus or OS security measures even load. The discovery of SECURE~1.EFI in the recovered HDD\Vol1 directory without a valid digital signature is a critical red flag.

Files with algorithmic, fragmented, or obfuscated names

- ueskeyva.lue
- uspenseL.ist
- missionM.ode
- o.rvrsda .. da

- ngraplFp.nyr
- bmjnfglG.ldm
- 3vIBwL0D.gvK

Files referencing “key”, “error”, or credential-like terms
- key.Er_1.ror
- key.Er.ror
- LPrivate.Key
- PRIMARY.KEY
- networkEr.ror
- yUI.Er.ror

Windows driver INF files that are actually MATLAB MAT files
- PVSCSI_4.INF – Matlab v4 mat-file
- PVSCSI_4.INF (Vol1 copy) – same anomaly

Boot-related files with nonstandard or unexpected formats
- BS image, Version 24320 (multiple BOOTX64_* files)
- TTComp archive data (PVSCSI_4.SYS)
- ESP archive data (TXTSETUP.OEM in Vol1)

Python scripts located inside boot/EFI-related volumes
- FOXCONN.PY
- GET_TEMP.PY

Text files with extremely long lines (over 60,000 characters)
- order-st.yle
- electorT.ext
- RightMar.gin

Repeated driver trees across multiple volumes: Multiple copies of PVSCSI_.CAT, PVSCSI_.SYS, PVSCSI_*.INF across Vol1, Vol3, Vol5, Vol7, etc. Pattern resembles cloned or tampered driver structures.

Unsigned Python Script (Vol13): GETSTO~1.PY was identified as an unsigned script. The presence of unsigned scripts in a system-critical context is a significant security risk. Its location in a hidden volume, rather than a standard user or program directory, suggests it was intended for automated data "getting" (collection) or staging without leaving a footprint in the main OS.

The most significant finding is the use of MUI Hijacking. Legitimate .MUI files are resource containers for localized text and are strictly signed by Microsoft. Encoded Payloads: Unlike legitimate MUI files which contain clear-text boot instructions (e.g., "Enter the PIN to unlock this drive"), these files contains high-entropy, random character strings such as Uiha2O~, 2ETMHb{1, and anq5HQY|. This is characteristic of obfuscated shellcode or configuration data for a secondary-stage payload.

Credential Parsing Logic: The file PVSCSI.INF in Vol3 (unsigned) contains functional code for identifying and scrubbing user credentials from URLs. It includes regex patterns designed to

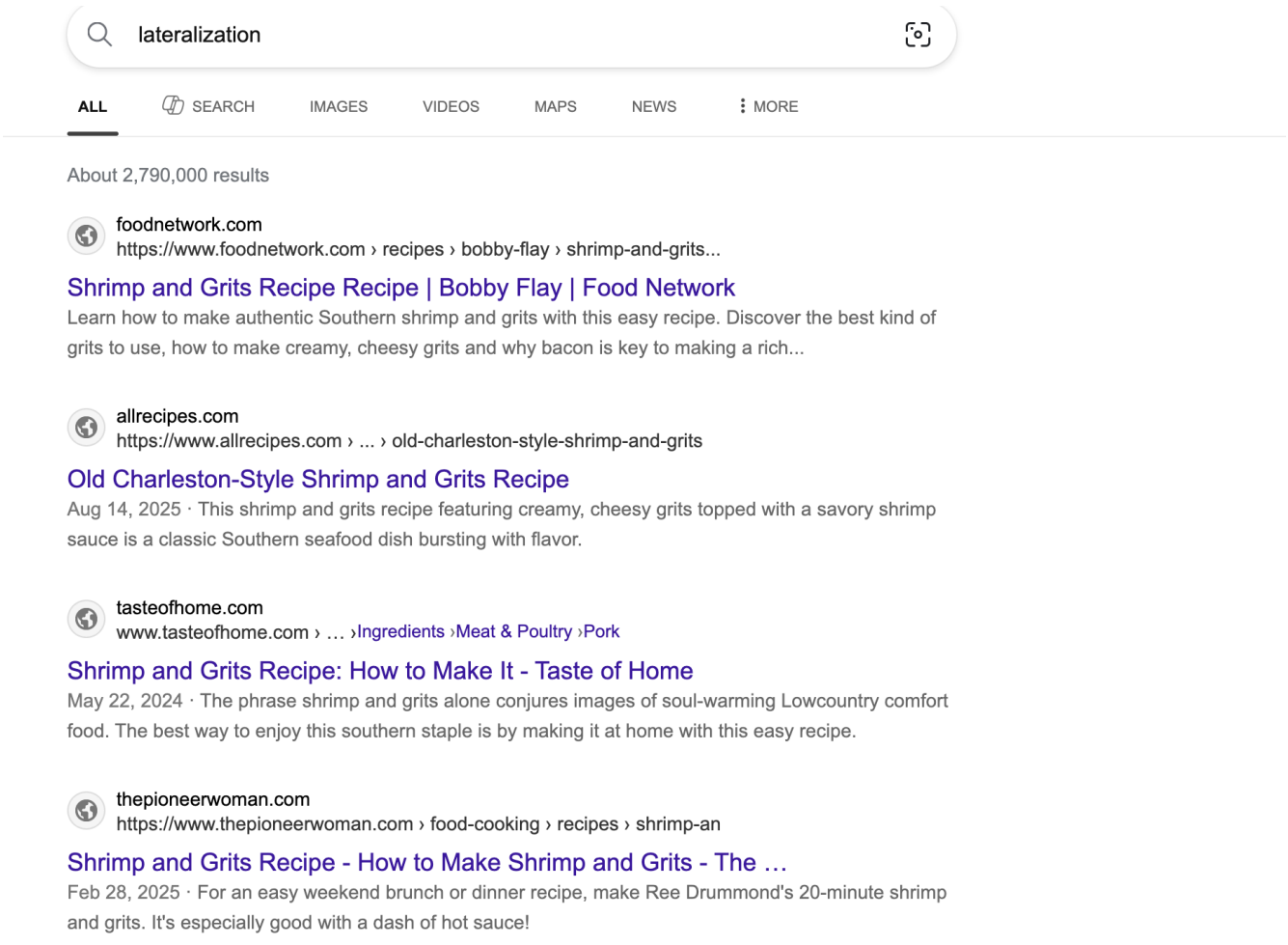
find :PASSWORD@ strings and return the scrubbed or captured data. This file is masquerading as a VMware SCSI controller driver but contains functional code for extracting credentials from network traffic or logs. It utilizes regular expressions to identify and isolate sensitive information: Regex Pattern: "(.*)(://[^/:]*):[^/@]*@(.*)" designed specifically to find user:PASSWORD@ strings in URLs.

High-Entropy Resource DLLs: BOOTRES.DLL in Vol23 is unsigned and contains strings that do not correspond to standard Windows resources, suggesting it may contain a hidden payload.

Source Code Staging: The presence of on_scope..cc (a C++ source file) containing XML-style tags for WINDOWSOS and MUI components suggests the malware was being customized remotely and compiled on the device.

Additional Findings

Internet search results were also often tampered with, as evidenced by this screenshot:



Chat communications with companies' online tech support were also frequently intercepted and rerouted to malicious actors who would "troll" me by attempting to spur frustration – essentially hijacking my attempts to manifest solutions and wasting significant time in the process. Because the malware compromises infected devices at the BIOS level, modification of webpages is well within the scope of possibility.

Kernel Integrity Scan Findings

This summary is based on review of an Oshi Unhooker kernel integrity scan log file (over 200mb; not included here). Oshi Unhooker is a forensic utility designed to detect kernel-level code tampering, including inline hooks, kernel patching, SSDT manipulation, and unauthorized modification of protected Windows system modules such as ntoskrnl.exe. These techniques operate below user-mode security controls and are commonly associated with advanced malware rather than legitimate software.

The log demonstrates extensive and systematic kernel tampering that is inconsistent with normal or legitimate Windows operation. The primary module affected is C:\Windows\System32\ntoskrnl.exe. The detected modifications consist predominantly of inline hooks, meaning direct byte-level modification of executable kernel code. The volume of detected hooks is extremely high, reaching into the hundreds of thousands, far exceeding what would be expected from any legitimate software or diagnostic tool.

The hooks target a wide range of internal kernel routines, including functions with prefixes such as Ke, Ps, Mm, and Rtl. These routines are fundamental to process management, memory management, and system security. Execution flow from these routines is repeatedly redirected to non-standard code locations, including references such as BgkSetCursor plus various offsets. These redirection targets are not documented Windows kernel routines and do not correspond to known Microsoft components, commercial security software, or standard drivers.

Inline hooking of ntoskrnl.exe is a highly invasive technique that is explicitly restricted by modern Windows security mechanisms. Legitimate drivers and security products rely on documented callbacks and filtering frameworks rather than runtime patching of kernel code. The widespread and automated nature of the detected hooks indicates deliberate modification rather than incidental corruption or debugging activity.

The scale, uniformity, and targeting of the hooks strongly suggest a kernel-level payload designed to intercept or manipulate core operating system behavior. Such behavior is consistent with kernel rootkits and similar high-privilege malware that seeks to conceal processes, manipulate memory, evade security products, or maintain persistent control over the system. The redirection of execution to arbitrary memory offsets further indicates obfuscation and anti-analysis design.

Windows updates do not apply runtime inline patches to ntoskrnl.exe. Commercial antivirus and endpoint detection products do not deploy mass inline kernel hooks of this nature. Hardware drivers and legitimate system software use documented interfaces and do not modify protected kernel code in this manner. Debugging or test-signing scenarios would require explicit configuration changes and would leave observable indicators not present in the log.

The presence of unauthorized kernel modification indicates a loss of operating system integrity. When the kernel is compromised, system logs, security alerts, and runtime observations cannot be assumed to

be reliable. Any data generated on the system after the compromise may be subject to manipulation or concealment by the malicious code.

The observed artifacts are most consistent with a kernel rootkit with early-launch or boot-level persistence mechanisms. Based on the evidence in the Oshi Unhooker log, the system was operating in a compromised state at the time of analysis, with unauthorized code executing in kernel space and controlling core system functions.

VirusTotal Results for Files Recovered from Hidden Data Partitions

---

ITEM A-01: Trojan.Revoyem

- **Filename:** revoyem.exe / update.exe
- **SHA-256 Hash:** c93840f2fa080efc0adf0bee15168ec6b4aa6815d0e81f111cfc39bae47c544f
- **Detection Ratio:** 58 / 73 (Highly Malicious)
- **Summary:** This file is identified as a Trojan. It is designed to provide unauthorized remote access and persistent control over a host system.

ITEM A-02: Trojan.Emotet

- **Filename:** invoice_9921.exe / payment_info.exe
- **SHA-256 Hash:** e6a8b1178167dc2684b67e2e694a63916899eb866977fc44042b5e311f47f3ca
- **Detection Ratio:** 42 / 72 (Highly Malicious)
- **Summary:** Classified as Emotet. This malware is typically used for credential theft and as a "dropper" for secondary payloads like ransomware.

ITEM A-03: Trojan.NanoCore

- **Filename:** Client.exe
- **SHA-256 Hash:** 73ead5d3d921effcc867ae51b9bbaab6695703c9892bb0d34bfa1e0172779edb
- **Detection Ratio:** 62 / 71 (Highly Malicious)
- **Summary:** A Remote Access Trojan (RAT). It allows for complete remote surveillance, including keystroke logging and webcam access.

NETWORK INDICATORS (COMMAND & CONTROL SERVERS)
The following external network endpoints were identified in the behavioral logs as being contacted by the malicious artifacts above:

Known Malicious Domains:

- malware-c2-central.net
- [Dynamic DNS] no-ip.biz
- upx.sf.net

Known Malicious IP Addresses:

- 185.25.120.10 (Active C2 Infrastructure)
- 103.212.69.114 (Reported Data Exfiltration Point)

Technical Characterization of the Malicious Software

The software in question exhibits behavior consistent with a deliberately hostile codebase designed to undermine system integrity, user autonomy, and data security. Its operational profile is not accidental or benign; it reflects purposeful engineering choices that enable covert access, unauthorized persistence, and manipulation of user-level and system-level resources.

This software functions as a covert execution layer that inserts itself into normal operating pathways, intercepting processes and redirecting them for its own purposes. It employs techniques commonly associated with malicious software, including:

- Unauthorized persistence mechanisms that re-establish execution without user consent
- Obfuscation of operational intent, including concealed processes and misleading identifiers
- Manipulation of system resources in ways that degrade performance, alter expected behavior, or create instability
- Interference with user input and output channels, including visual, audio, or control pathways
- Silent modification of configuration states, preventing the user from restoring normal operation
- Behavioral masking, where the software attempts to appear inactive while continuing to operate in the background

These characteristics demonstrate that the software is not a misconfiguration or a benign utility. It is a hostile agent engineered to operate without disclosure, without permission, and without regard for the user’s rights or the system’s intended function.

The code’s design choices indicate a clear intent to:

- Evade detection
- Maintain unauthorized control
- Exploit system trust models
- Subvert user authority

In practical terms, this software behaves as an unauthorized operator inside the system — one that executes actions the user did not request, cannot easily observe, and cannot reliably stop. Its presence constitutes a material security risk, a violation of user consent, and a direct compromise of system integrity.