COS20030 Assignment 2

Core Task Group Report

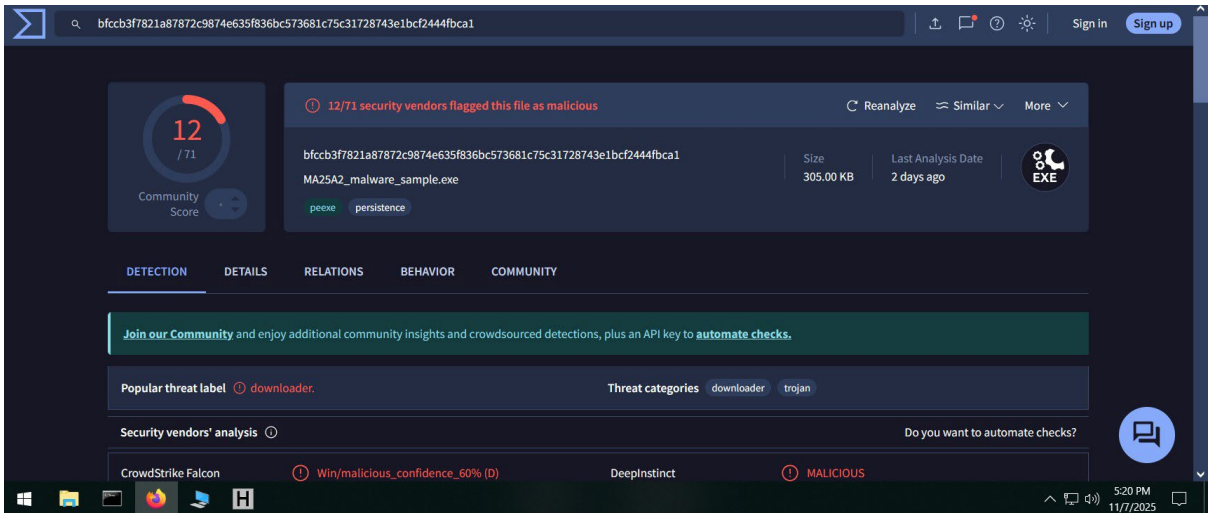
Group Members:

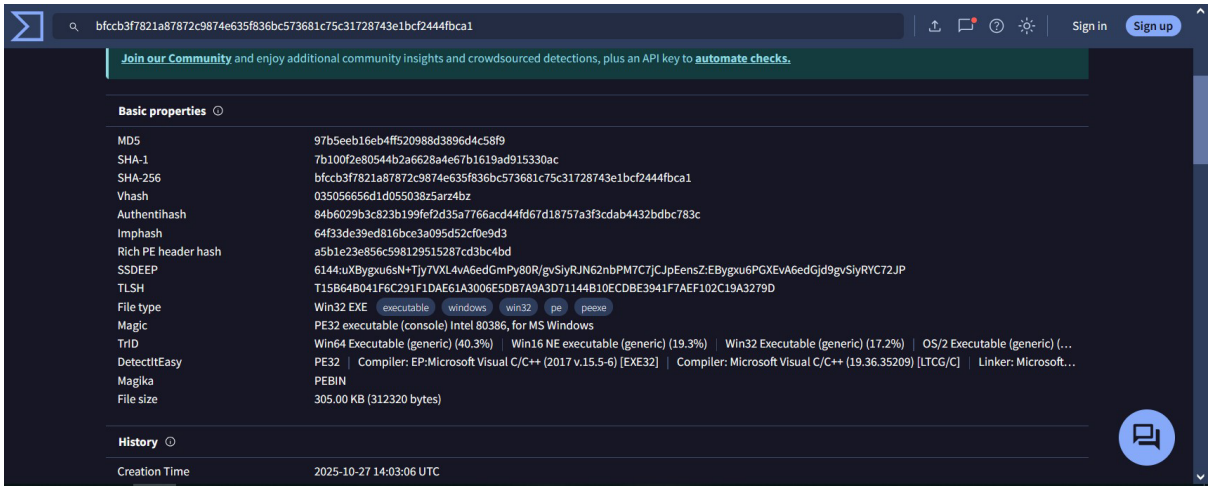
Name	Student ID
Clement Chang Cheng	102774912
Phang Xia Hui	102773508

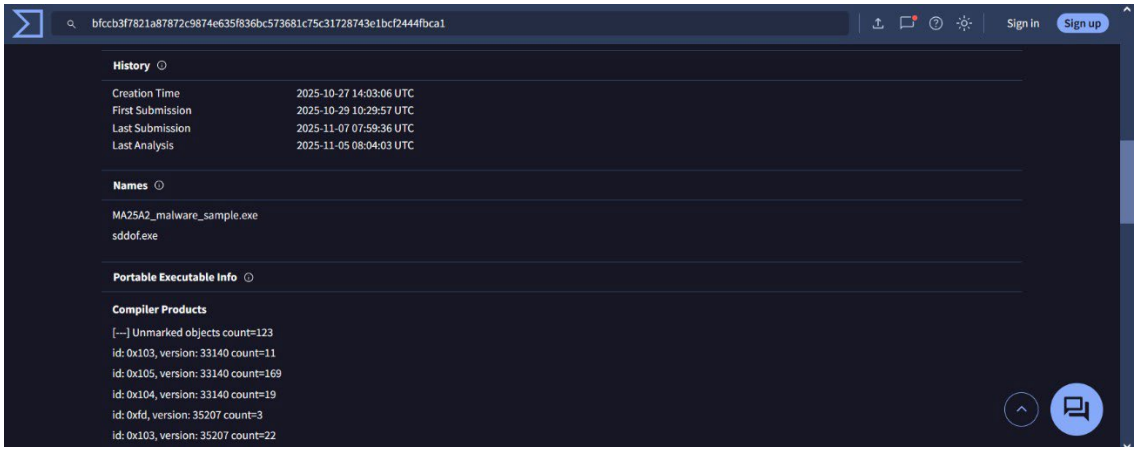
1.0 Basic Static Analysis Results

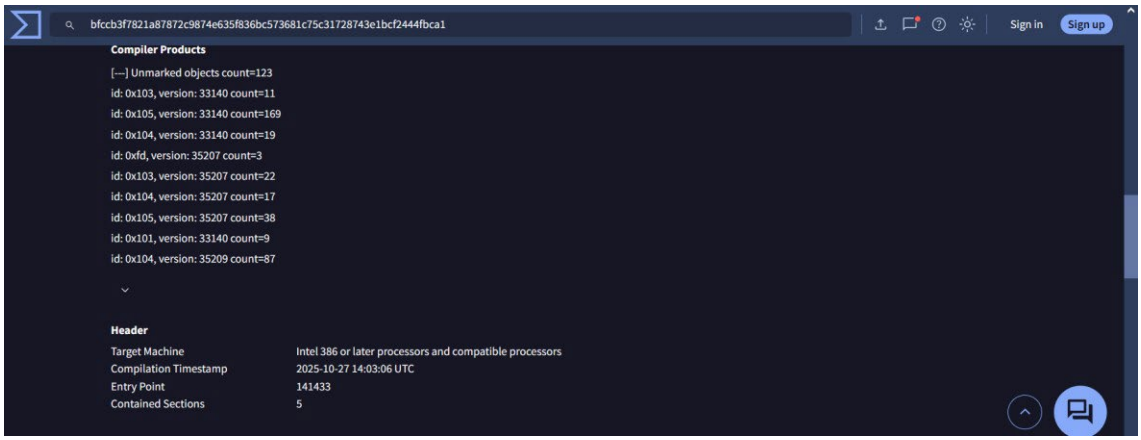
The initial phase of the investigation involved Basic Static Analysis to gather preliminary intelligence on the suspect binary MA25A2_malware_sample.exe. This process focused on identifying the file's properties, cryptographic hashes, and potential indicators of malicious intent without executing the code.

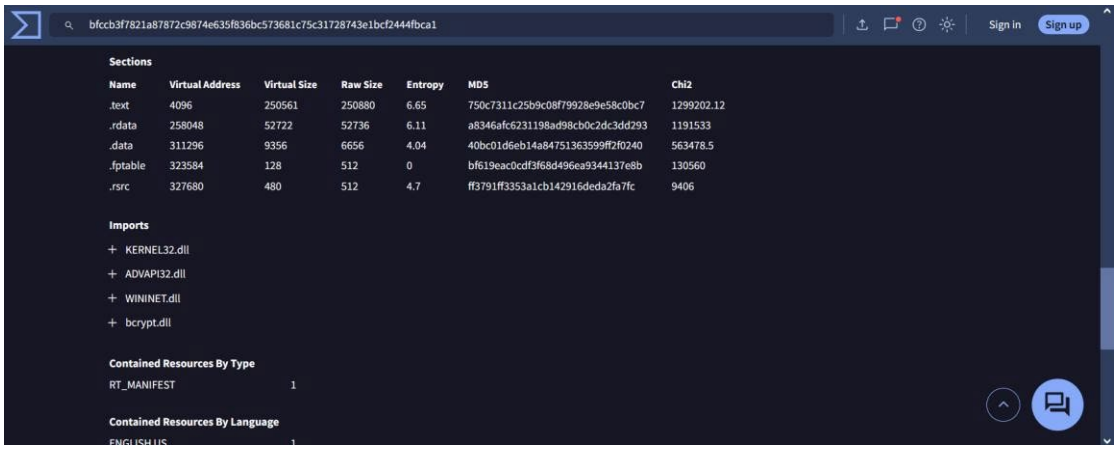
1.1 VirusTotal Intelligence

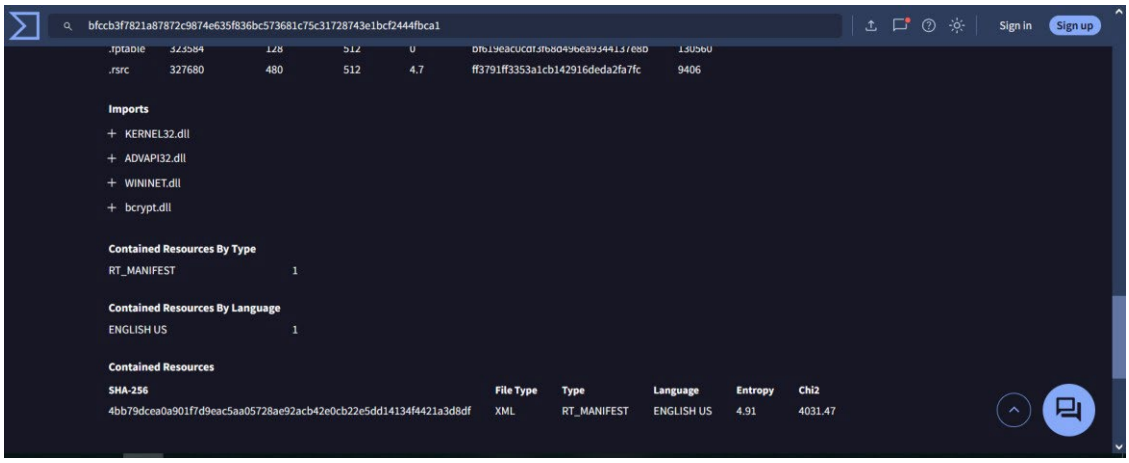



bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1
Sign in
Sign up
Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.
Basic properties
MD5 97b5eeb16eb4ff520988d3896d4c58f9
SHA-1 7b100f2e80544b2a6628a4e67b1619ad915330ac
SHA-256 bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1
Vhash 035056656d1d055038z5arz4bz
Authentihash 84b6029b3c823b199fef2d35a7766acd44fd67d18757a3f3cdab4432bdbc783c
Imphash 64f33de39ed816bce3a095d52cf0e9d3
Rich PE header hash a5b1e23e856c598129515287cd3bc4bd
SSDEEP 6144:uXBygxu6sN+Tjy7VXL4vA6edGmPy80R/gvSiyRJN62nbPM7C7jCJpEensZ:EBygxu6PGXEvA6edGjd9gvSiyRYC72JP
TLSH T15B64B041F6C291F1DAE61A3006E5DB7A9A3D71144B10ECDBE3941F7AEF102C19A3279D
File type Win32 EXE executable windows win32 pe peexe
Magic PE32 executable (console) Intel 80386, for MS Windows
TrID Win64 Executable (generic) (40.3%) | Win16 NE executable (generic) (19.3%) | Win32 Executable (generic) (17.2%) | OS/2 Executable (generic) (...
DetectItEasy PE32 | Compiler: EP:Microsoft Visual C/C++ (2017 v.15.5-6) [EXE32] | Compiler: Microsoft Visual C/C++ (19.36.35209) [LTCG/C] | Linker: Microsoft...
Magika PEBIN
File size 305.00 KB (312320 bytes)
History
Creation Time 2025-10-27 14:03:06 UTC


bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1
Sign in
Sign up
History
Creation Time 2025-10-27 14:03:06 UTC
First Submission 2025-10-29 10:29:57 UTC
Last Submission 2025-11-07 07:59:36 UTC
Last Analysis 2025-11-05 08:04:03 UTC
Names
MA25A2_malware_sample.exe
sddof.exe
Portable Executable Info
Compiler Products
[---] Unmarked objects count=123
id: 0x103, version: 33140 count=11
id: 0x105, version: 33140 count=169
id: 0x104, version: 33140 count=19
id: 0xfd, version: 35207 count=3
id: 0x103, version: 35207 count=22


bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1
Sign in
Sign up
Compiler Products
[---] Unmarked objects count=123
id: 0x103, version: 33140 count=11
id: 0x105, version: 33140 count=169
id: 0x104, version: 33140 count=19
id: 0xfd, version: 35207 count=3
id: 0x103, version: 35207 count=22
id: 0x104, version: 35207 count=17
id: 0x105, version: 35207 count=38
id: 0x101, version: 33140 count=9
id: 0x104, version: 35209 count=87
Header
Target Machine Intel 386 or later processors and compatible processors
Compilation Timestamp 2025-10-27 14:03:06 UTC
Entry Point 141433
Contained Sections 5


bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1
Sign in
Sign up
Sections
Name Virtual Address Virtual Size Raw Size Entropy MD5 Chi2
.text 4096 250561 250880 6.65 750c7311c25b9c08f79928e9e58c0bc7 1299202.12
.rdata 258048 52722 52736 6.11 a8346afc6231198ad98cb0c2dc3dd293 1191533
.data 311296 9356 6656 4.04 40bc01d6eb14a84751363599ff2f0240 563478.5
.fptable 323584 128 512 0 bf619eac0cdf3f68d496ea9344137e8b 130560
.rsrc 327680 480 512 4.7 ff3791ff3353a1cb142916deda2fa7fc 9406
Imports
KERNEL32.dll
ADVAPI32.dll
WININET.dll
bcrypt.dll
Contained Resources By Type
RT_MANIFEST 1
Contained Resources By Language


bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1
Sign in
Sign up
.rsrc 327680 480 512 4.7 ff3791ff3353a1cb142916deda2fa7fc 9406
Imports
KERNEL32.dll
ADVAPI32.dll
WININET.dll
bcrypt.dll
Contained Resources By Type
RT_MANIFEST 1
Contained Resources By Language
ENGLISH US 1
Contained Resources
SHA-256 File Type Type Language Entropy Chi2
4bb79dcea0a901f7d9eac5aa05728ae92acb42e0cb22e5dd14134f4421a3d8df XML RT_MANIFEST ENGLISH US 4.91 4031.47

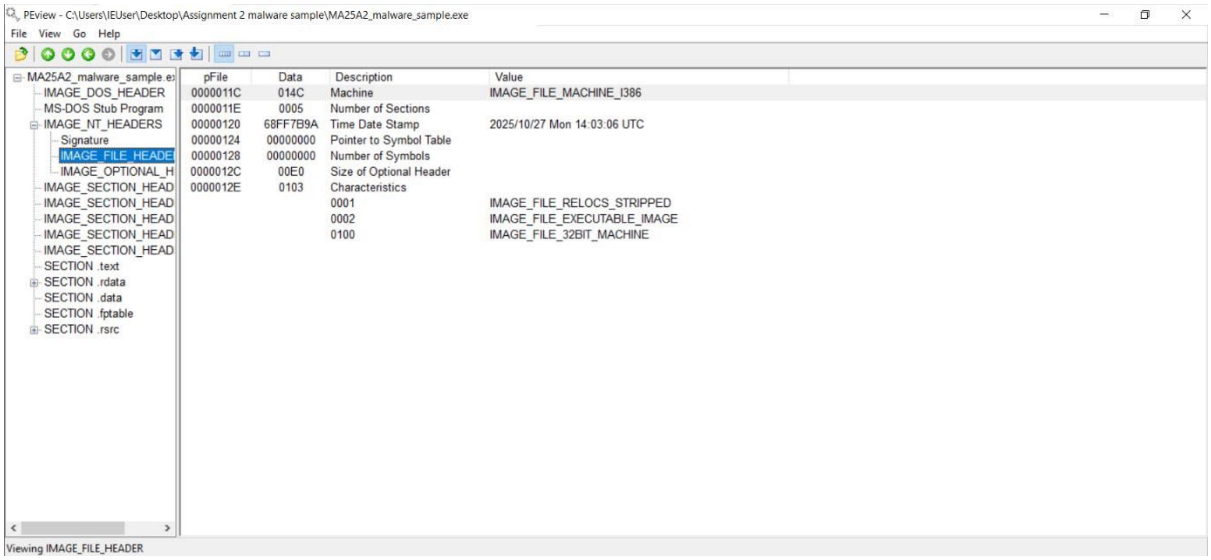
The sample was submitted to VirusTotal to check for known signatures and community reputation.

- **Detection Rate:** The file was flagged as malicious by **12 out of 71** security vendors.
- **Classification:** The threat is primarily categorized as a **Trojan** and **Downloader**. This suggests the malware's primary function is not necessarily to cause immediate damage itself, but to connect to a remote server and download additional malicious payloads.
- **File Hashes:**
    - **MD5:** 97b5eeb16eb4ff520988d3896d4c58f9
    - **SHA-256:** bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1

Analysis: The detection ratio (12/71) is relatively low for a known malware sample. This, combined with the "Downloader" classification, often indicates that the file is a new variant or a "dropper" specifically designed to evade signature-based detection before fetching its main payload.

1.2 PEView and Timestamp Anomalies

We examined the Portable Executable (PE) header using PEView to inspect the file's metadata and compilation artifacts.



Compilation Timestamp: 2025/10/27 Mon 14:03:06 UTC.

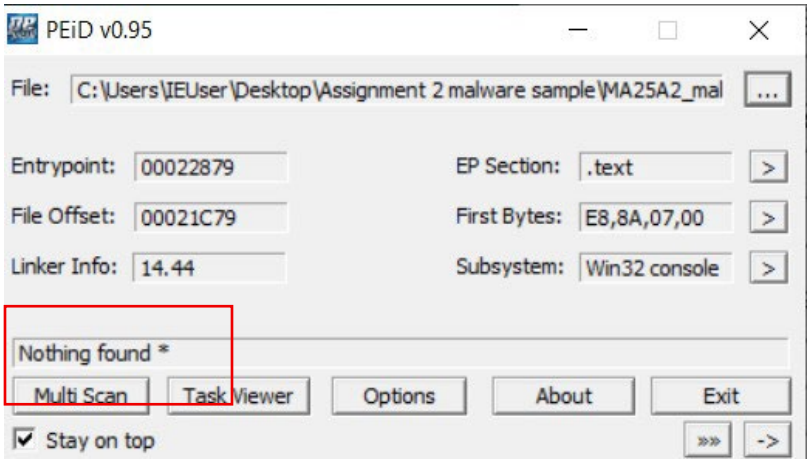
Observation: The analysis date (Nov 7, 2025) suggests this file was compiled recently relative to the current date, or potentially *in the future* depending on the exact analysis timeframe.

Analysis: Malware authors frequently manipulate the Time Date Stamp field in the PE header via "timestomping". By setting the compilation date to a future or seemingly legitimate past

date, attackers attempt to blend in with legitimate system files and evade forensic timelines or YARA rules that filter based on file age.

1.3 Packer Identification (PEiD)

To determine if the malware was obfuscated, we utilized PEiD.

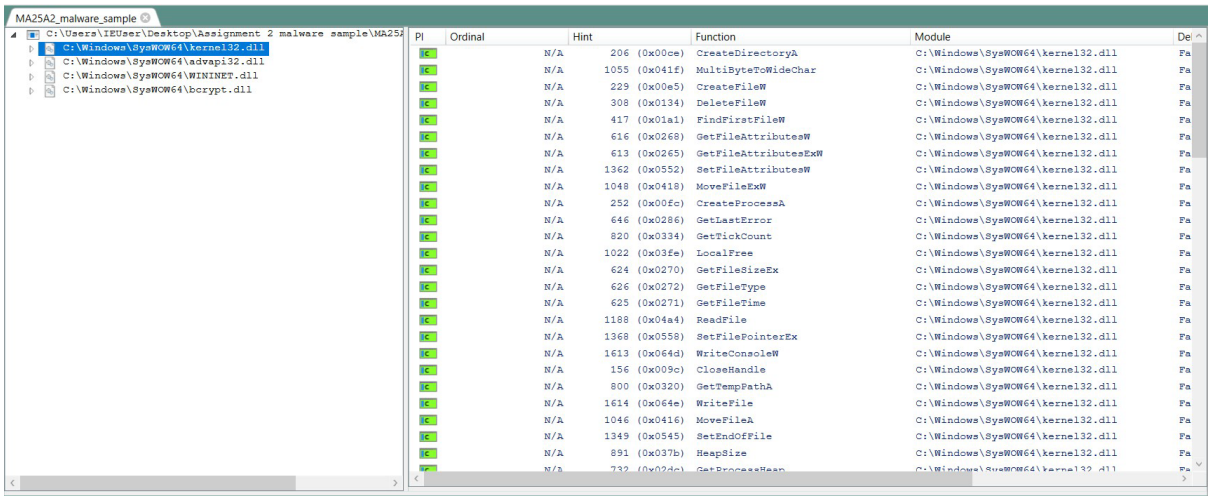


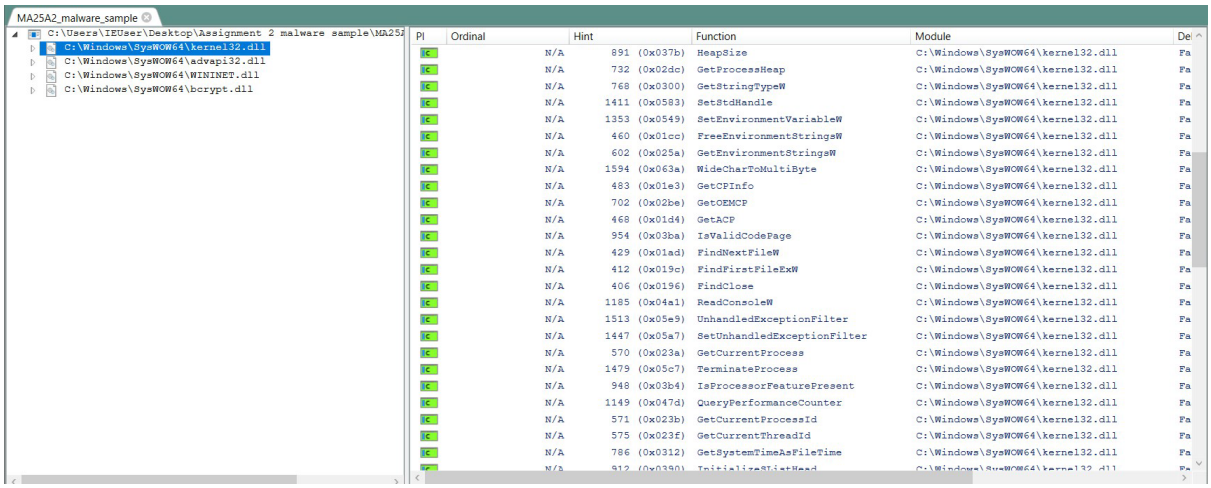
PEiD Result: The tool returned **"Nothing found *"**.

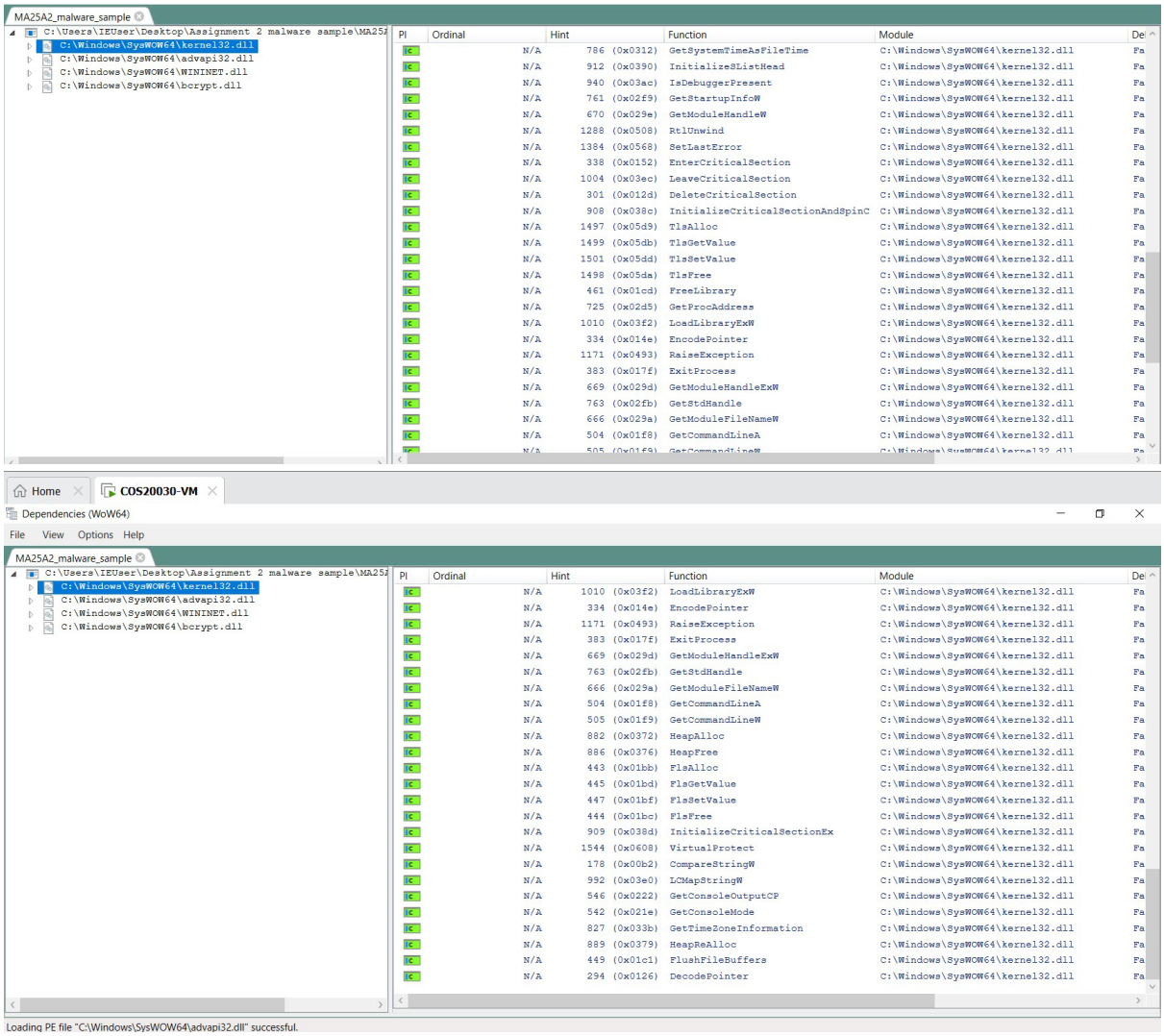
Analysis: The "Nothing found" result in PEiD is a strong indicator of a custom or unknown crypter. The malware code is compressed or encrypted to hide its true instructions from static analysis tools. This requires dynamic unpacking to reveal its true capabilities.

1.4 Dependency Analysis (Dependency Walker)

Dependency Walker was used to list the Dynamic Link Libraries (DLLs) and functions imported by the malware. The imports provided a high-level overview of the malware's capabilities, although many were likely hidden by the packer.


MA25A2_malware_sample
C:\Users\IEUser\Desktop\Assignment 2 malware sample\MA25
C:\Windows\SysWOW64\kernel32.dll
C:\Windows\SysWOW64\advapi32.dll
C:\Windows\SysWOW64\WININET.dll
C:\Windows\SysWOW64\bcrypt.dll
PI Ordinal Hint Function Module De
N/A 206 (0x00ce) CreateDirectoryA C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1055 (0x041f) MultiByteToWideChar C:\Windows\SysWOW64\kernel32.dll Fa
N/A 229 (0x00e5) CreateFileW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 308 (0x0134) DeleteFileW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 417 (0x01a1) FindFirstFileW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 616 (0x0268) GetFileAttributesW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 613 (0x0265) GetFileAttributesExW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1362 (0x0552) SetFileAttributesW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1048 (0x0418) MoveFileExW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 252 (0x00fc) CreateProcessA C:\Windows\SysWOW64\kernel32.dll Fa
N/A 646 (0x0286) GetLastError C:\Windows\SysWOW64\kernel32.dll Fa
N/A 820 (0x0334) GetTickCount C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1022 (0x03fe) LocalFree C:\Windows\SysWOW64\kernel32.dll Fa
N/A 624 (0x0270) GetFileSizeEx C:\Windows\SysWOW64\kernel32.dll Fa
N/A 626 (0x0272) GetFileType C:\Windows\SysWOW64\kernel32.dll Fa
N/A 625 (0x0271) GetFileTime C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1188 (0x04a4) ReadFile C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1368 (0x0558) SetFilePointerEx C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1613 (0x064d) WriteConsoleW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 156 (0x009c) CloseHandle C:\Windows\SysWOW64\kernel32.dll Fa
N/A 800 (0x0320) GetTempPathA C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1614 (0x064e) WriteFile C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1046 (0x0416) MoveFileA C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1349 (0x0545) SetEndOfFile C:\Windows\SysWOW64\kernel32.dll Fa
N/A 891 (0x037b) HeapSize C:\Windows\SysWOW64\kernel32.dll Fa

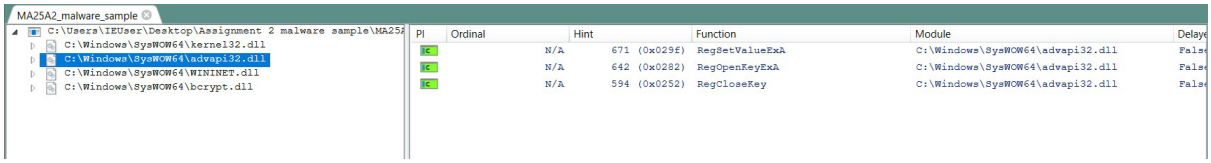

MA25A2_malware_sample
C:\Users\IEUser\Desktop\Assignment 2 malware sample\MA25
C:\Windows\SysWOW64\kernel32.dll
C:\Windows\SysWOW64\advapi32.dll
C:\Windows\SysWOW64\WININET.dll
C:\Windows\SysWOW64\bcrypt.dll
PI Ordinal Hint Function Module De
N/A 891 (0x037b) HeapSize C:\Windows\SysWOW64\kernel32.dll Fa
N/A 732 (0x02dc) GetProcessHeap C:\Windows\SysWOW64\kernel32.dll Fa
N/A 768 (0x0300) GetStringTypeW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1411 (0x0583) SetStdHandle C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1353 (0x0549) SetEnvironmentVariableW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 460 (0x01cc) FreeEnvironmentStringsW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 602 (0x025a) GetEnvironmentStringsW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1594 (0x063a) WideCharToMultiByte C:\Windows\SysWOW64\kernel32.dll Fa
N/A 483 (0x01e3) GetCPInfo C:\Windows\SysWOW64\kernel32.dll Fa
N/A 702 (0x02be) GetOEMCP C:\Windows\SysWOW64\kernel32.dll Fa
N/A 468 (0x01d4) GetACP C:\Windows\SysWOW64\kernel32.dll Fa
N/A 954 (0x03ba) IsValidCodePage C:\Windows\SysWOW64\kernel32.dll Fa
N/A 429 (0x01ad) FindNextFileW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 412 (0x019c) FindFirstFileExW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 406 (0x0196) FindClose C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1185 (0x04a1) ReadConsoleW C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1513 (0x05e9) UnhandledExceptionFilter C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1447 (0x05a7) SetUnhandledExceptionFilter C:\Windows\SysWOW64\kernel32.dll Fa
N/A 570 (0x023a) GetCurrentProcess C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1479 (0x05c7) TerminateProcess C:\Windows\SysWOW64\kernel32.dll Fa
N/A 948 (0x03b4) IsProcessorFeaturePresent C:\Windows\SysWOW64\kernel32.dll Fa
N/A 1149 (0x047d) QueryPerformanceCounter C:\Windows\SysWOW64\kernel32.dll Fa
N/A 571 (0x023b) GetCurrentProcessId C:\Windows\SysWOW64\kernel32.dll Fa
N/A 575 (0x023f) GetCurrentThreadId C:\Windows\SysWOW64\kernel32.dll Fa
N/A 786 (0x0312) GetSystemTimeAsFileTime C:\Windows\SysWOW64\kernel32.dll Fa



Library import: kernel32:dll

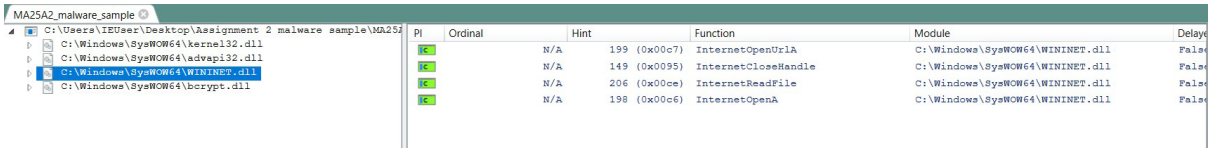
Functions import: CreateDirectoryA, MultiByteToWideChar, CreateFileW, DeleteFileW, FindFirstFileW, GetFileAttributesW, GetFileAttributesExW, SetFileAttributesW, MoveFileExW, CreateProcessA, GetLastError, GetTickCount, LocalFree, GetFileSizeEx, GetFileType, GetFileTime, ReadFile, SetFilePointerEx, WriteConsoleW, CloseHandle, GetTempPathA, WriteFile, MoveFileA, SetEndOfFile, HeapSize, GetProcessHeap, GetStringTypeW, SetStdHandle, SetEnvironmentVariableW, FreeEnvironmentStringsW, GetEnvironmentStringsW, WideCharToMultiByte, GetCPInfo, GetOEMCP, GetACP, IsValidCodePage, FindNextFileW, FindFirstFileExW, FindClose, ReadConsoleW, UnhandledExceptionFilter, SetUnhandledExceptionFilter, GetCurrentProcess, TerminateProcess, IsProcessFeaturePresent, QueryPerformanceCounter, GetCurrentProcessId, GetCurrentThreadId, GetSystemTimeAsFileTime, InitializeSListHead, IsDebuggerPresent, GetStartupInfoW, GetModuleHandleW, RtlUnwind, SetLastError, EnterCriticalSection, LeaveCriticalSection, DeleteCriticalSection, InitializeCriticalSectionAndSpinC, TlsAlloc, TlsGetValue, TlsSetValue, TlsFree, FreeLibrary, GetProcAddress, LoadLibraryExW, EncodePointer, RaiseException, ExitProcess, GetModuleHandleExW, GetStdHandle,

GetModuleFileNameW, GetCommandLineA, GetCommandLineW, HeapAlloc, HeapFree, FlsAlloc, FlsGetValue, FlsSetValue, FlsFree, InitializeCriticalSectionEx, VirtualProtect, CompareStringW, LCMapStringW, GetConsoleOutputCP, GetConsoleMode, GetTimeZoneInformation, HeapReAlloc, FlushFileBuffers, DecodePointer



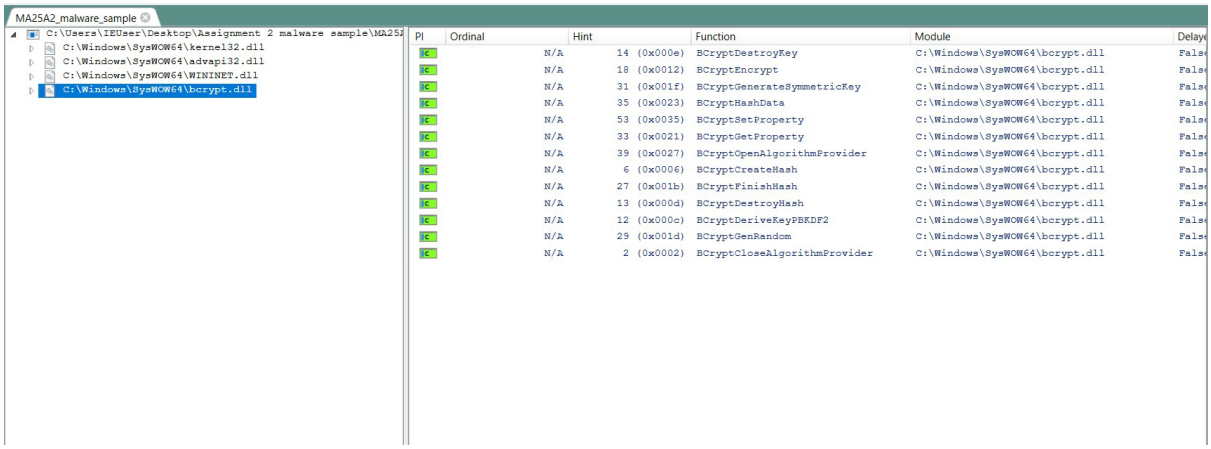
Library imports: advapi32.dll

Functions import: RegSetValueExA, RegOpenKeyExA, RegCloseKey



Library imports: WININET.dll

Functions import: InternetOpenUrlA, InternetCloseHandle, InternetReadFile, InternetOpenA



Library imports: bcrypt.dll

Functions: BCryptDestroKey, BCryptEncrypt, BCryptGenerateSymmetricKey, BCryptHashData, BCryptSetProperty, BCryptGetProperty, BCryptOpenAlgorithmProvider, BCryptCreateHash, BCryptFinishHash, BCryptDestroyHash, BCryptDeriveKeyPBKDF2, BCryptGenRandom, BCryptCloseAlgorithmProvider

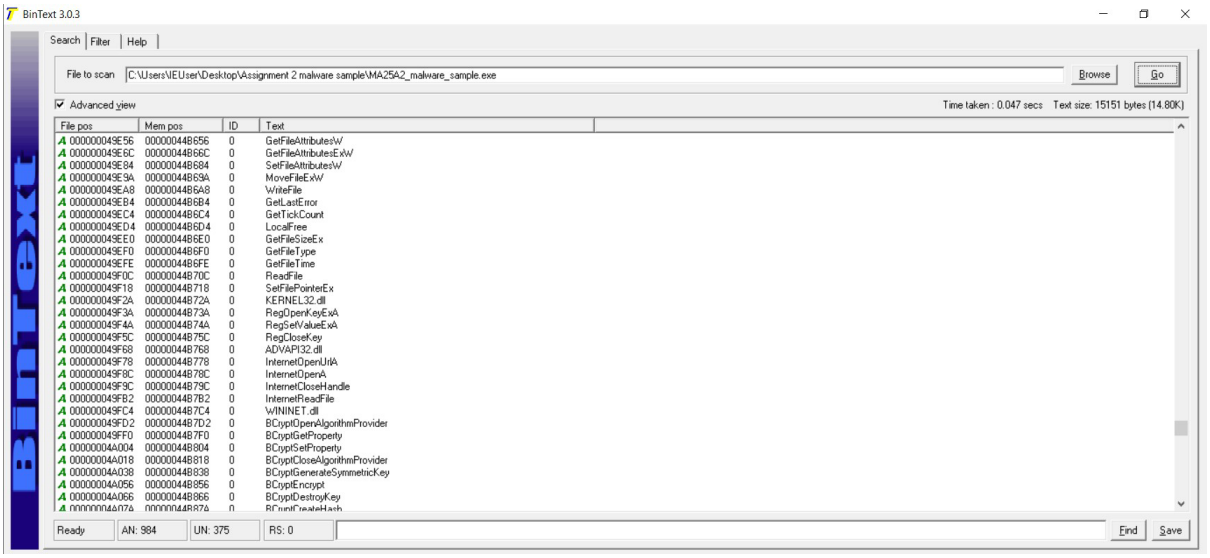
Key Imported DLLs:

- **WININET.dll:** Imports InternetOpenUrlA and InternetReadFile. This confirms the malware has the capability to initiate network connections and download data from the internet, aligning with the "Downloader" classification.

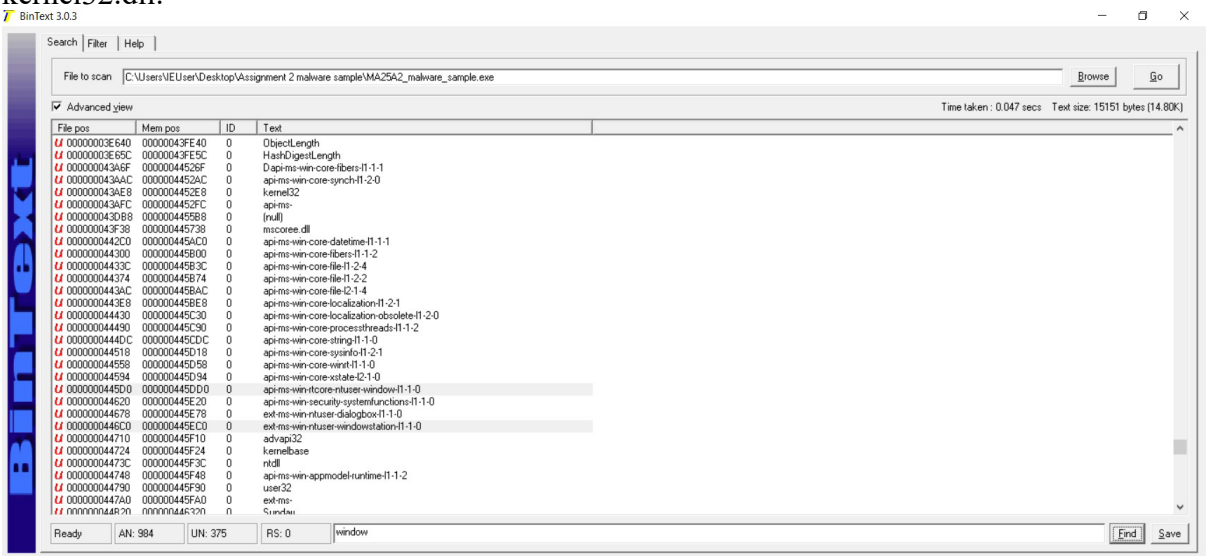
- **ADVAPI32.dll:** Imports RegSetValueExA and RegOpenKeyExA. These functions are used to modify the Windows Registry, strongly suggesting the malware attempts to achieve **persistence** (surviving system reboots).
- **BCRYPT.dll:** Imports various encryption functions like BCryptDecrypt and BCryptEncrypt. This indicates the malware likely uses encryption, possibly to decrypt its downloaded payload or to secure its command-and-control (C2) communications.
- **KERNEL32.dll:** Imports standard file manipulation APIs (CreateFile, WriteFile), further supporting the hypothesis that it drops files onto the disk.

1.5 String Analysis (BinText)

We scanned the binary for readable strings to find hardcoded IP addresses, URLs, or messages.



BinText revealed the all-caps fake DLL KERNEL32.dll at offset 0x4F52A which is a classic DLL search-order hijacking IOC that executes malicious code before the legitimate system kernel32.dll.



These strings are automatically injected by the crypter when it replaces direct kernel32.dll imports with delayed Universal CRT redirects to hide real API calls until runtime.

Findings: The initial string scan revealed very little intelligible text. A search for keywords like "github" returned zero results in the packed file.

Analysis: The lack of readable strings is consistent with the file being packed. The packer has obfuscated the sensitive data.

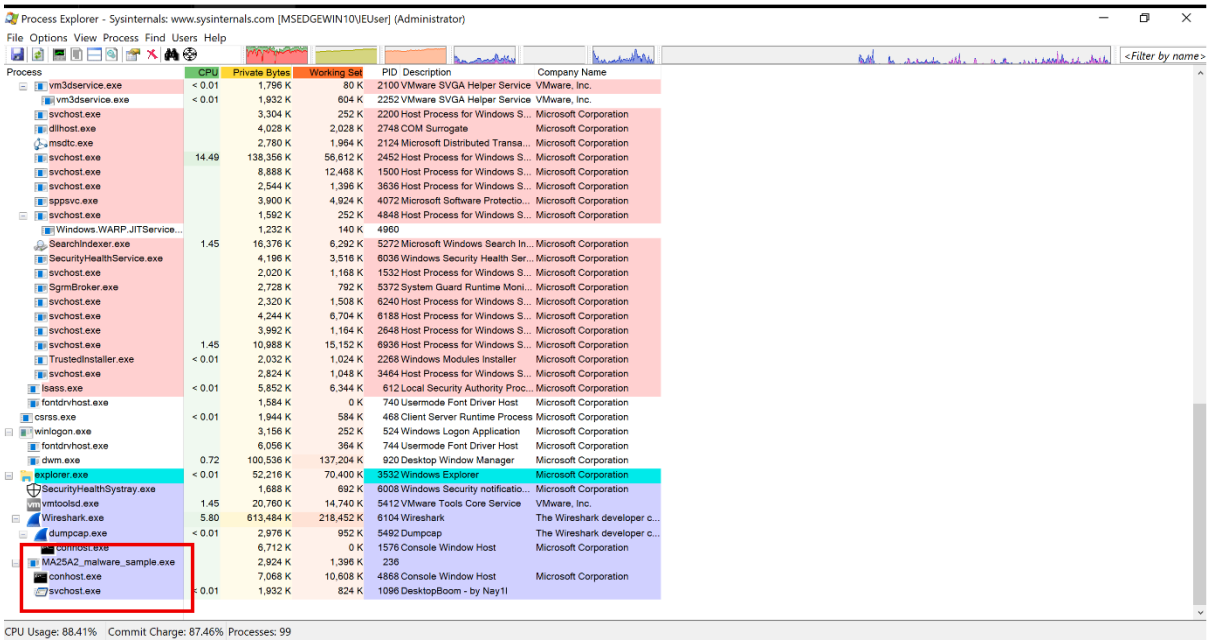
Fake DLL Indicator: A closer inspection revealed a string referencing a fake DLL path: C:\Windows\SysWOW64\kernel32.dll. This is a potential Indicator of Compromise (IOC) for **DLL Search Order Hijacking**, where the malware attempts to trick the operating system into loading a malicious DLL instead of the legitimate system library.

2.0 Basic Dynamic Analysis Results

During the basic dynamic analysis phase, the sample MA25A2_malware_sample.exe was executed within a controlled virtual environment. The analysis focused on observing the malware's interactions with the file system, registry, and network in real-time using Process Monitor, Process Explorer, and Wireshark. The specimen demonstrated behaviour consistent with evasion and reconnaissance.

2.1 Execution

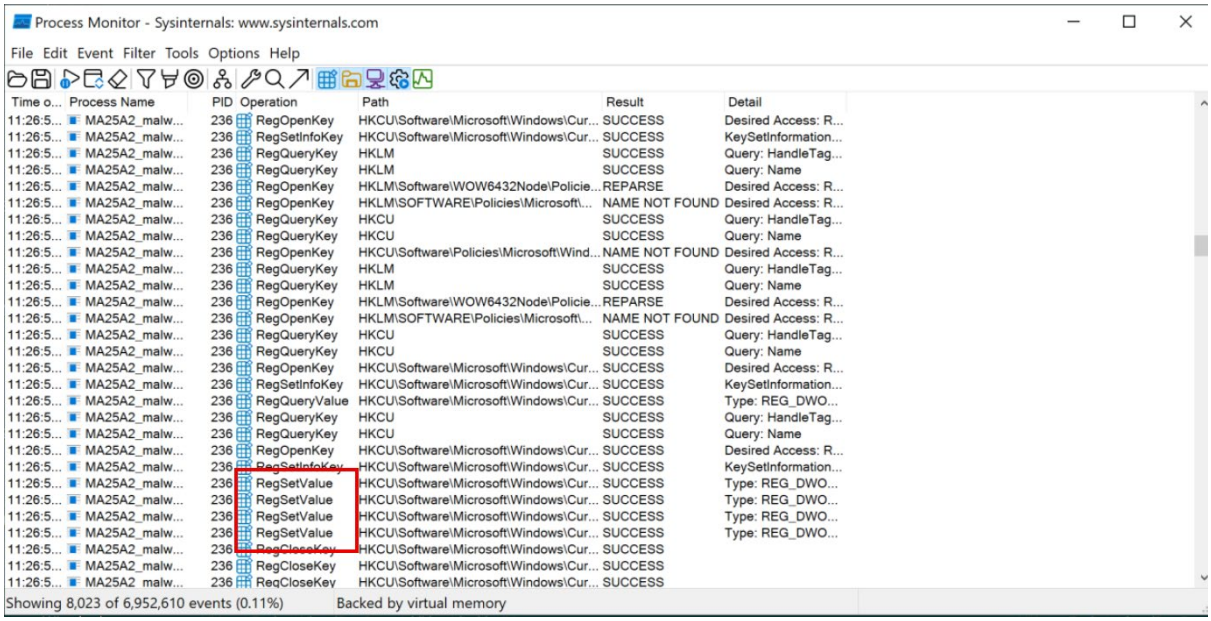
Upon executing the sample, we first examined the process tree in **Process Explorer** to understand how the malware launched.



As observed in the screenshot above, MA25A2_malware_sample.exe spawned child processes, specifically conhost.exe and svchost.exe. Legitimate svchost.exe processes are typically children of services.exe. The fact that this malware launched svchost.exe as its own child process indicates process impersonation. This is a common evasion technique designed to hide malicious activity behind a benign-looking system process name.

2.2 Registry Alteration

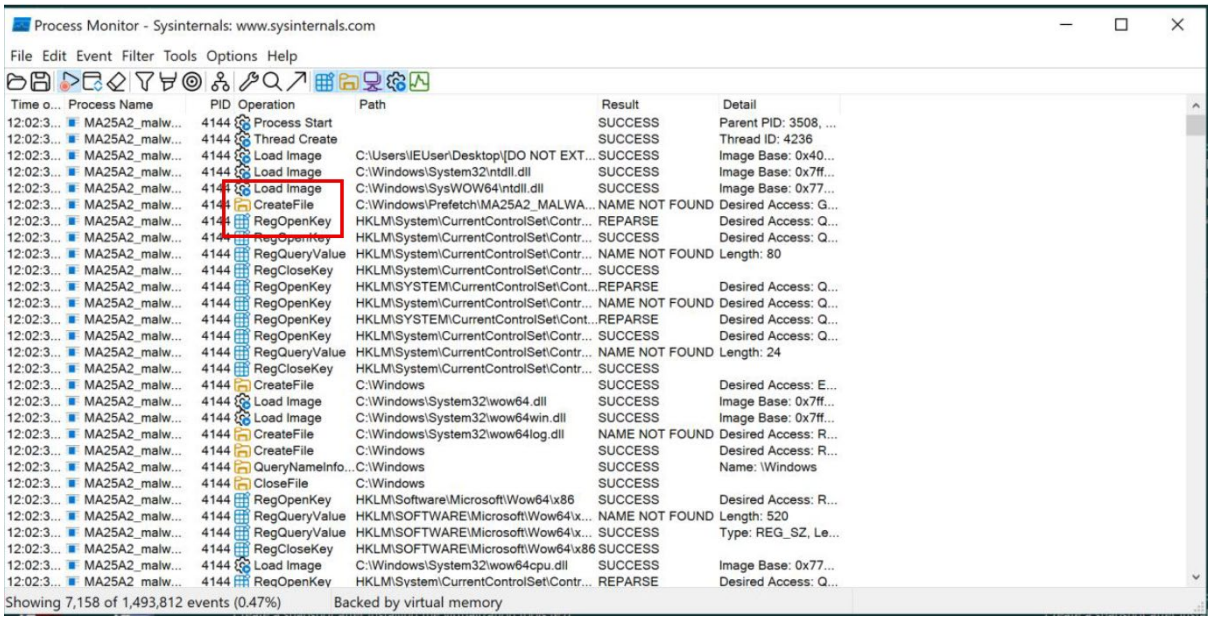
Next, we analysed **Process Monitor** events to identify changes to the system configuration:



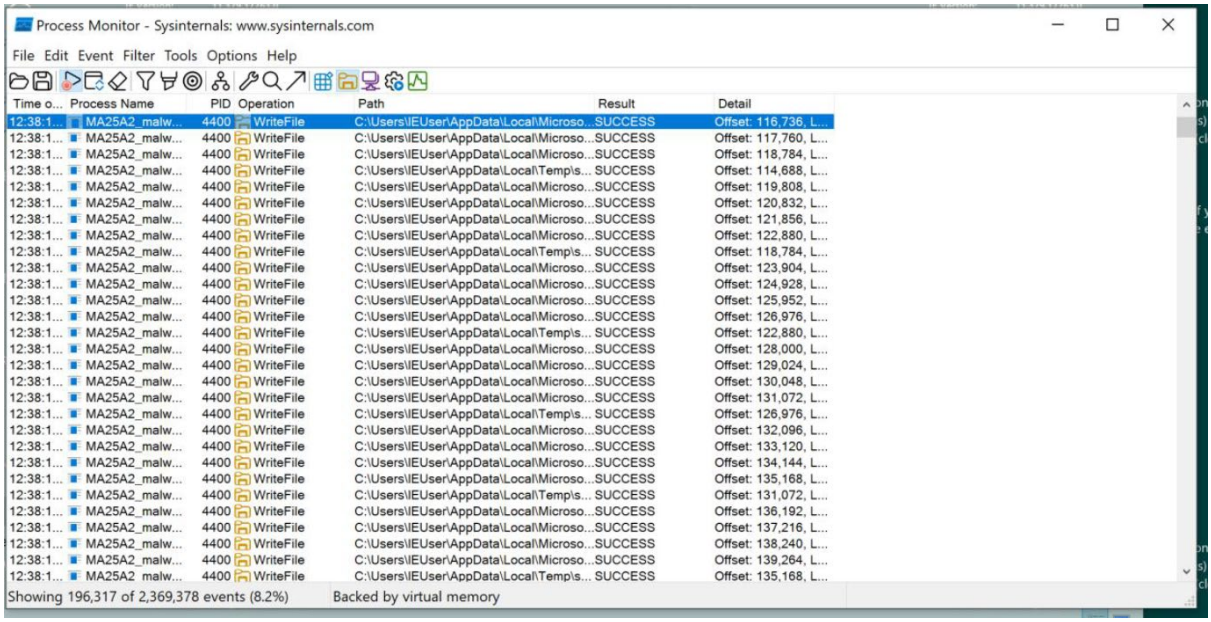
The capture above highlights multiple ***RegSetValue*** operations. The malware successfully wrote data to the registry keys under HKCU\Software\Microsoft\Windows\CurrentVersion.... Modifying these keys often indicates the malware is storing configuration settings or attempting to establish **persistence**, ensuring it remains active or re-launches after a system reboot.

2.3 File System Alteration

The malware demonstrated significant interaction with the file system. We tracked the sequence of file creation and data writing using **Process Monitor**.



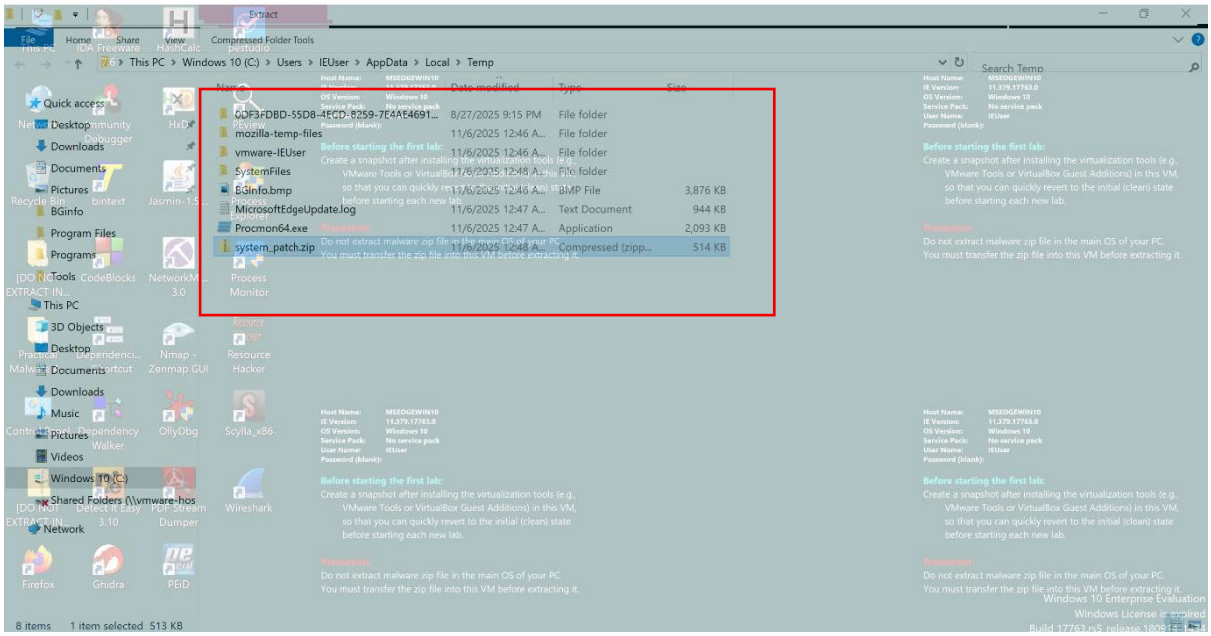
Initial ***CreateFile*** events as shown above revealed the malware obtaining handles to files in the **C:\\Windows\Prefetch\\MA25A2_MALWARE…..** directory. This was immediately followed by data ingestion:

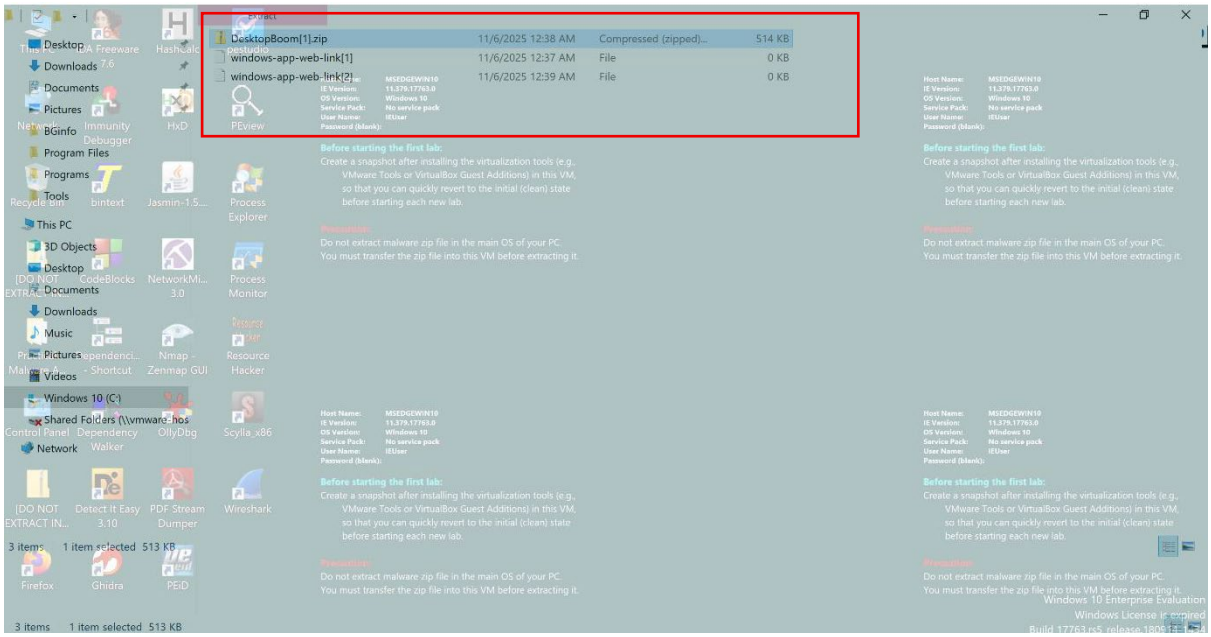


The ***WriteFile*** events confirmed that the malware was actively writing payloads to disk. Specifically, it targeted:

- C:\Users\IEUser\AppData\Local\Temp
- C:\Users\IEUser\AppData\Local\Microsoft\Windows\INetCache\IE\1BQLVC6W

To verify these findings, we navigated to the directories in the VM:

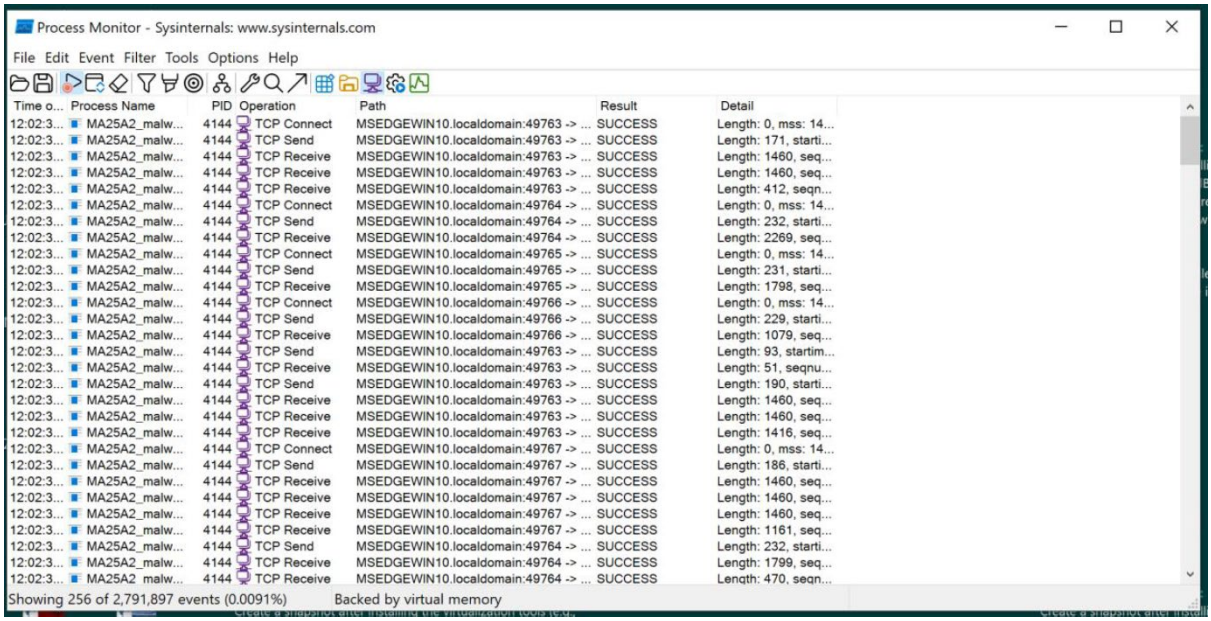




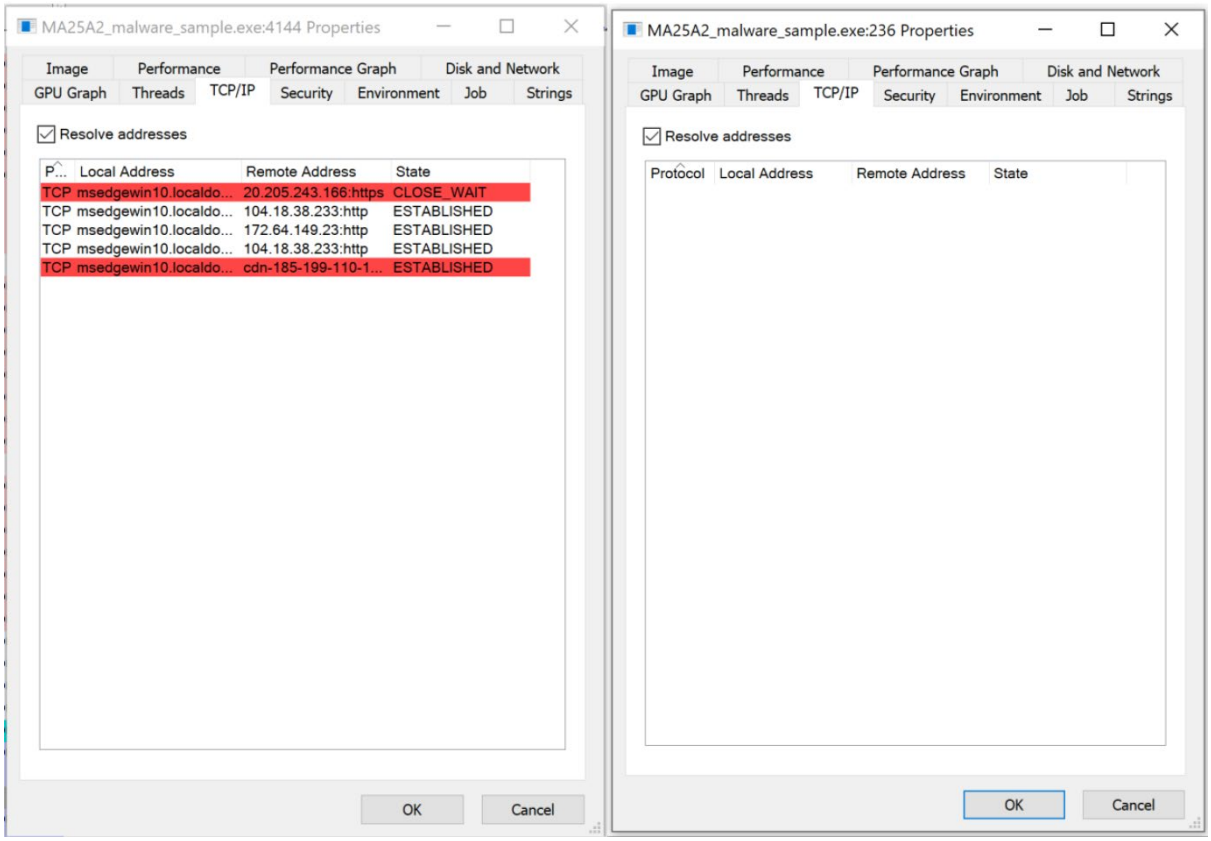
The screenshots above confirm the successful dropping of these artifacts such as the **DesktopBoom zip file** that would suggest desktop related activities due to its name. The use of the Temp and INetCache folders would suggest directories writable by standard users, allowing the malware to operate without needing administrative privileges. All the while the file explorer starts experiencing screen shaking and visibility issues.

2.4 Network Analysis and Reconnaissance

Finally, we investigated the network traffic generated by the malware.

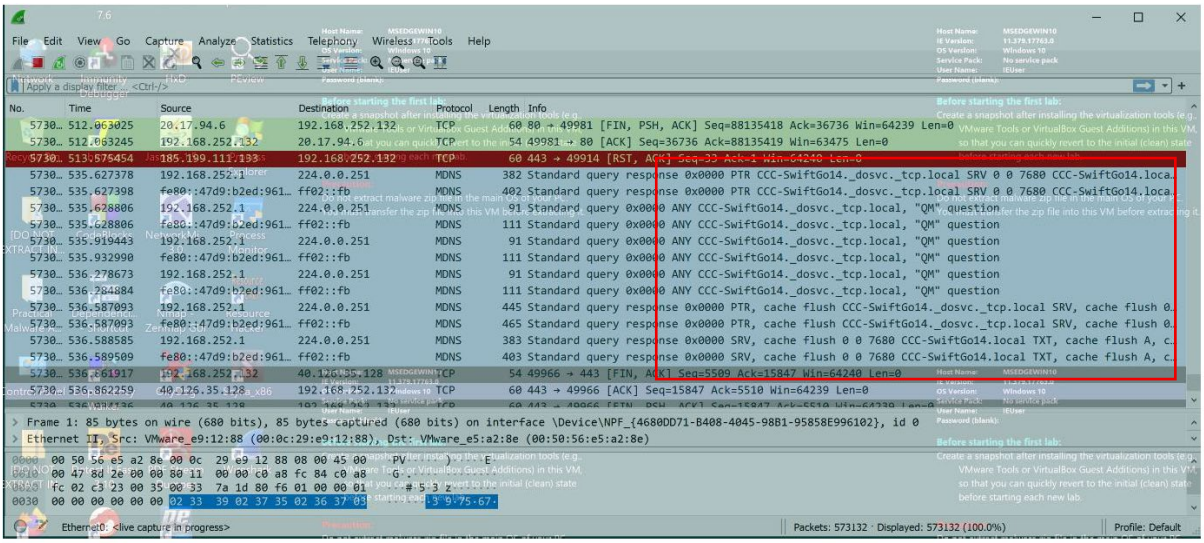


Process Monitor as shown above first alerted us to TCP activity, showing the process actively connecting to and receiving data from external endpoints. To get a clearer view of the destinations, we checked the TCP/IP tab in **Process Explorer**:



The properties window revealed established connections to external IP addresses, such as 104.18.38.233 and 20.205.243.166 which immediately disappears after. This suggests command-and-control (C2) communication or payload downloading.

To analyse the content of the traffic, we turned to **Wireshark**:



The packet capture uncovered unusual Multicast DNS or mDNS activity. The malware generated **multiple ANY requests** querying for **_dosv._tcp.local**. This is a potential indicator of network discovery. The malware is likely scanning the local subnet to identify other vulnerable hosts or specific services to propagate to, similarly to worms.

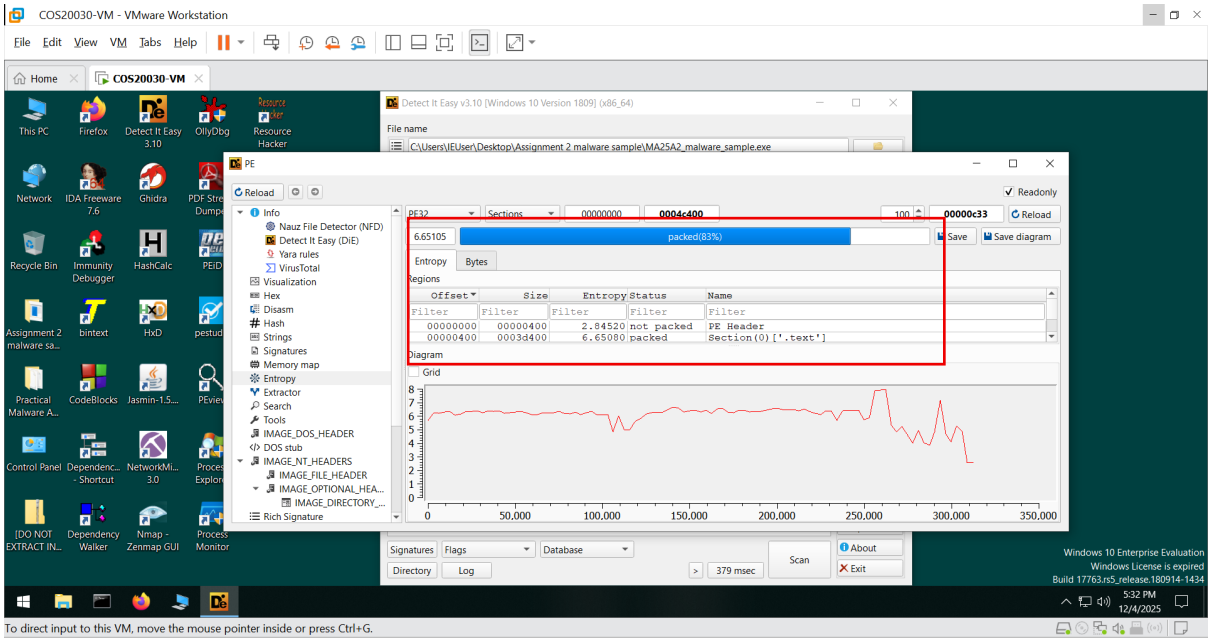
This Basic dynamic analysis confirms that **MA25A2_malware_sample.exe** is a sophisticated threat. It utilizes process impersonation to blend in, **drops persistence artifacts** in user-writable directories, and **performs active network reconnaissance** to potentially spreads itself across the network.

3.0 Advanced Static Analysis Results

Following the basic analysis, the team performed advanced static analysis to dissect the malware's internal code structure and capabilities without execution. This phase utilized **Detect It Easy (DIE)** for packer identification, **IDA Pro** for disassembly, and **Capa** for automated capability detection.

3.1 Packer Identification and Entropy Analysis

The malware sample was first analysed using **Detect It Easy (DIE)** to examine its entropy levels and identify potential packing or obfuscation.



Observation: The analysis revealed a high entropy value of **6.65** for the .text section whilst indicating the packed percentage of **83%**. The entropy graph consistently hovered in the 6-7 range, which is characteristic of compressed or encrypted data.

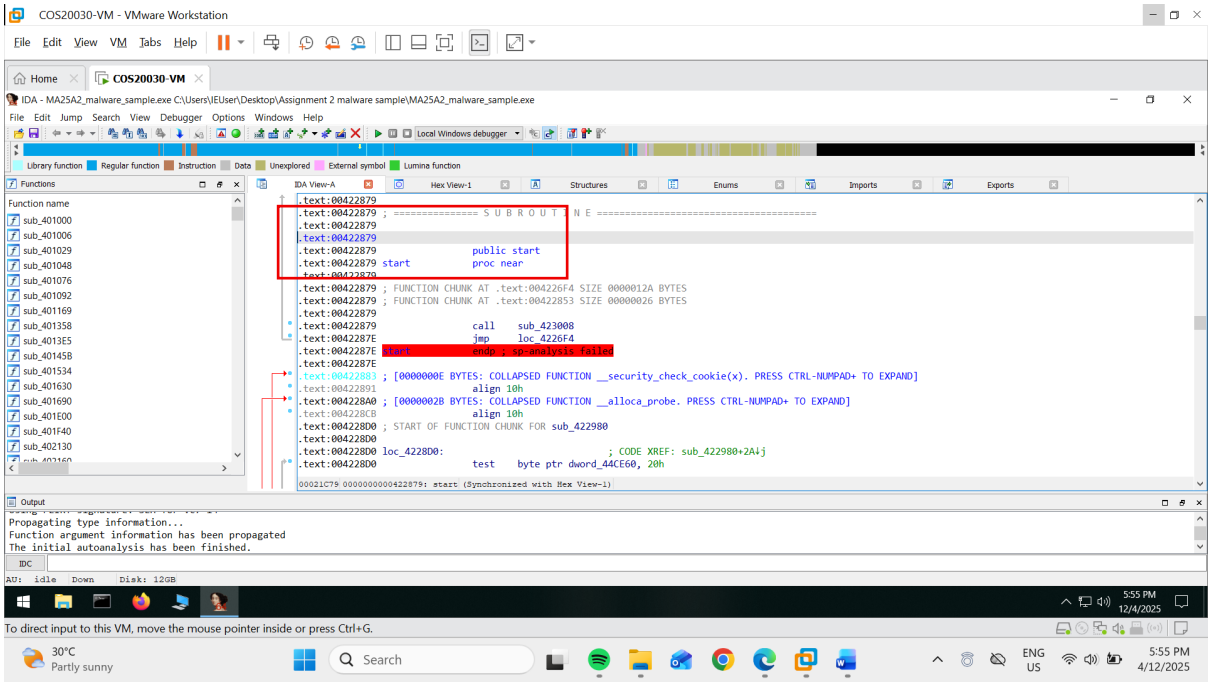
Analysis: A high entropy value is a strong indicator that the malware author has employed packing techniques to compress or encrypt the executable's code. This is a primary evasion tactic designed to conceal the malware's true functionality from antivirus scanners and hinder reverse engineering efforts. This finding necessitated the dynamic unpacking process detailed in the "Advanced Dynamic Analysis" section.

3.2 Disassembly and Code Analysis (IDA Pro)

The sample was loaded into **IDA Pro** to inspect its assembly code and internal logic. This deep dive revealed critical insights into the malware's entry point, obfuscation methods, and persistence mechanisms.

3.2.1 Entry Point Identification

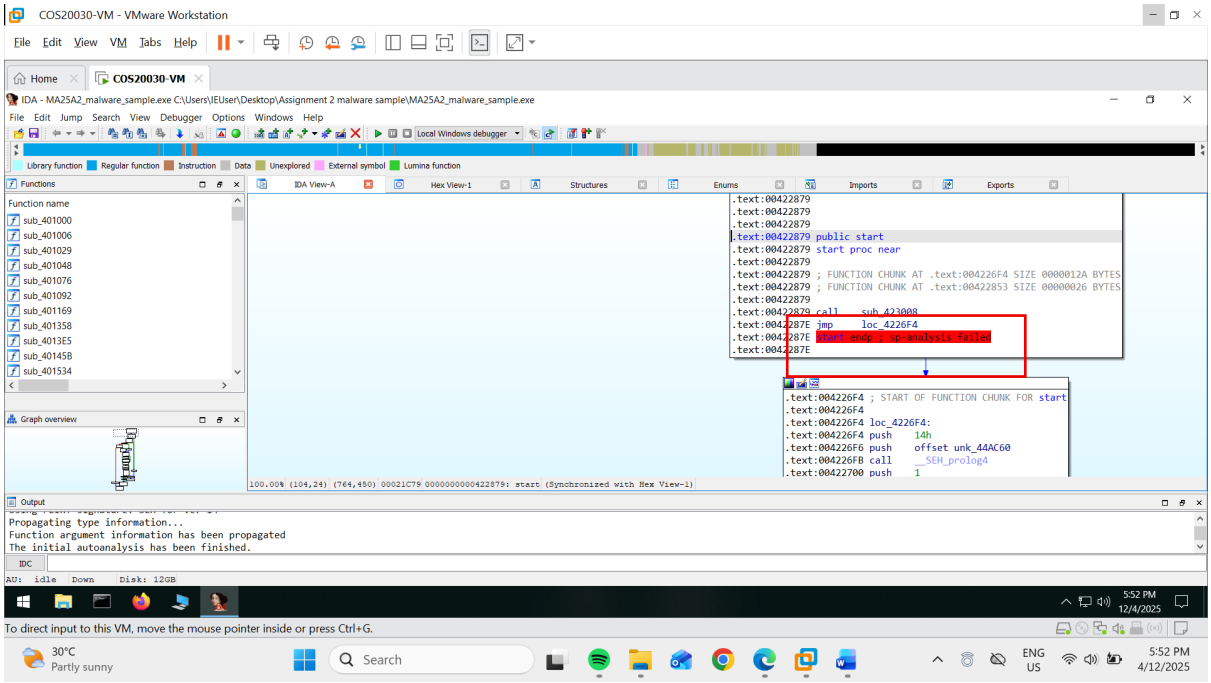
Upon loading the binary, the program's Entry Point (EP) was identified at address 0x00422879. This is the memory address where the operating system transfers control to start the malware's execution.



Observation: The disassembly window clearly displays the start function at 0x00422879. The code begins with a call to a subroutine sub_423008 followed by a jump instruction, which is typical of standard Windows executable initialization.

3.2.2 Obfuscation Indicators (Stack Analysis Error)

Immediately following the entry point instructions, IDA Pro flagged a critical analysis warning.

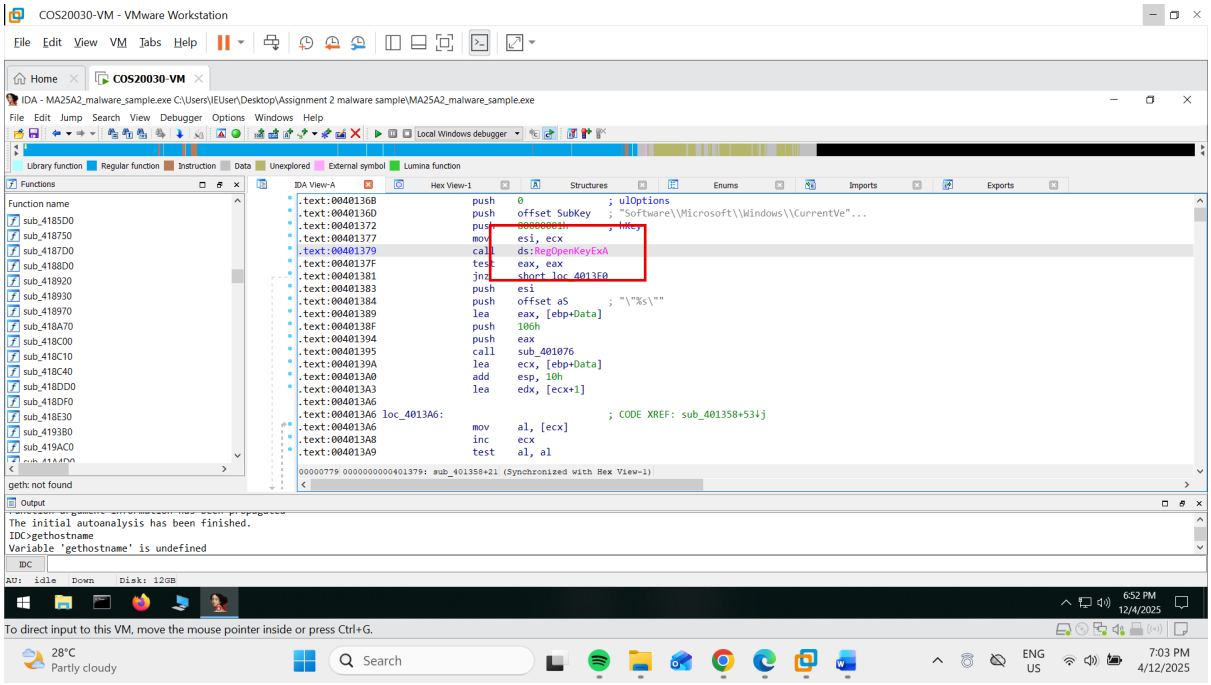


Observation: The disassembler displayed the error message: **"sp-analysis failed"** (Stack Pointer analysis failed) at the end of the start function.

Analysis: This error indicates that the malware performs non-standard manipulations of the stack pointer during its initialization. This is a common **anti-disassembly technique** used by malware authors to confuse static analysis tools. By altering the stack pointer in unexpected ways, the malware attempts to prevent automated tools from correctly mapping the function's structure, thereby hindering reverse engineering efforts.

3.2.3 Identification of Persistence Mechanism

A primary objective of the analysis was to determine how the malware survives system reboots. By inspecting the imports from ADVAPI32.dll, we traced the usage of the RegOpenKeyExA API to discover the persistence logic.

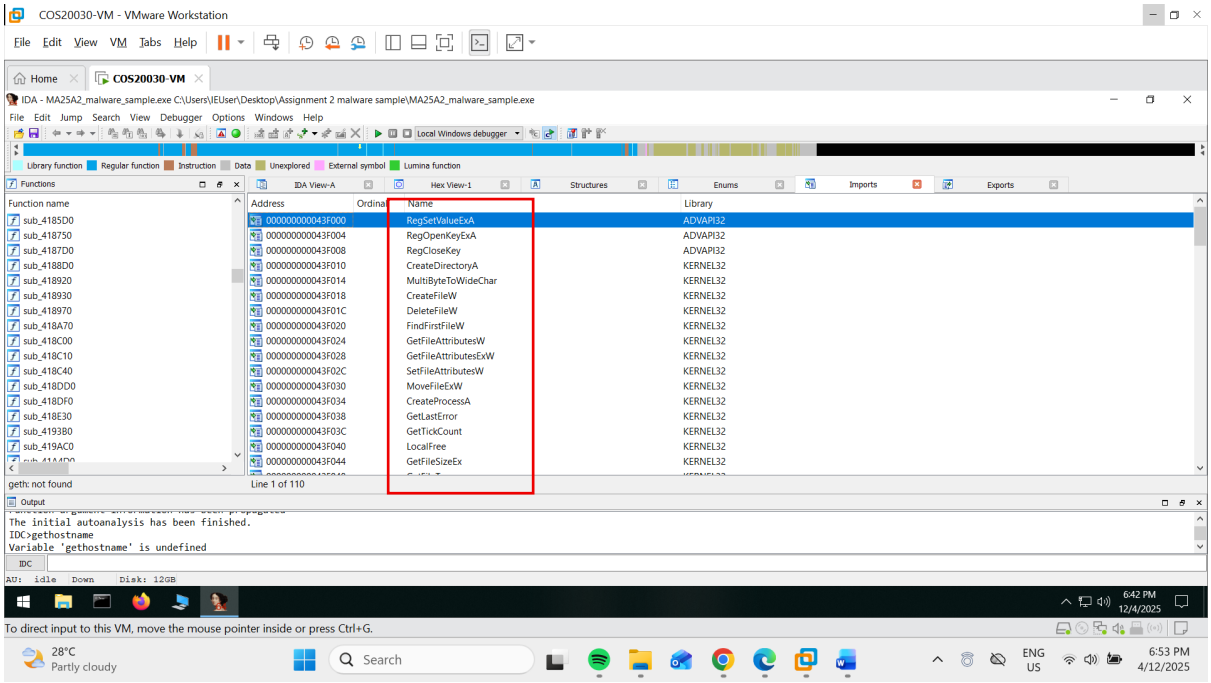


Observation: Tracing the code to address 0x00401379 revealed the "smoking gun." The malware explicitly pushes the string **"Software\Microsoft\Windows\CurrentVersion\Run"** onto the stack as an argument for the RegOpenKeyExA function call.

Analysis: This confirms that the malware achieves **persistence** by adding an entry to the Windows "Run" registry key. This specific key is a well-known Autostart Extensibility Point (ASEP). Any program listed in this key will execute automatically every time the user logs in, ensuring the malware maintains long-term access to the infected system.

3.2.4 File System Manipulation Capabilities (General)

To understand the malware's interaction with the host system, we examined the Imported Address Table (IAT) for file-related APIs imported from KERNEL32.dll.



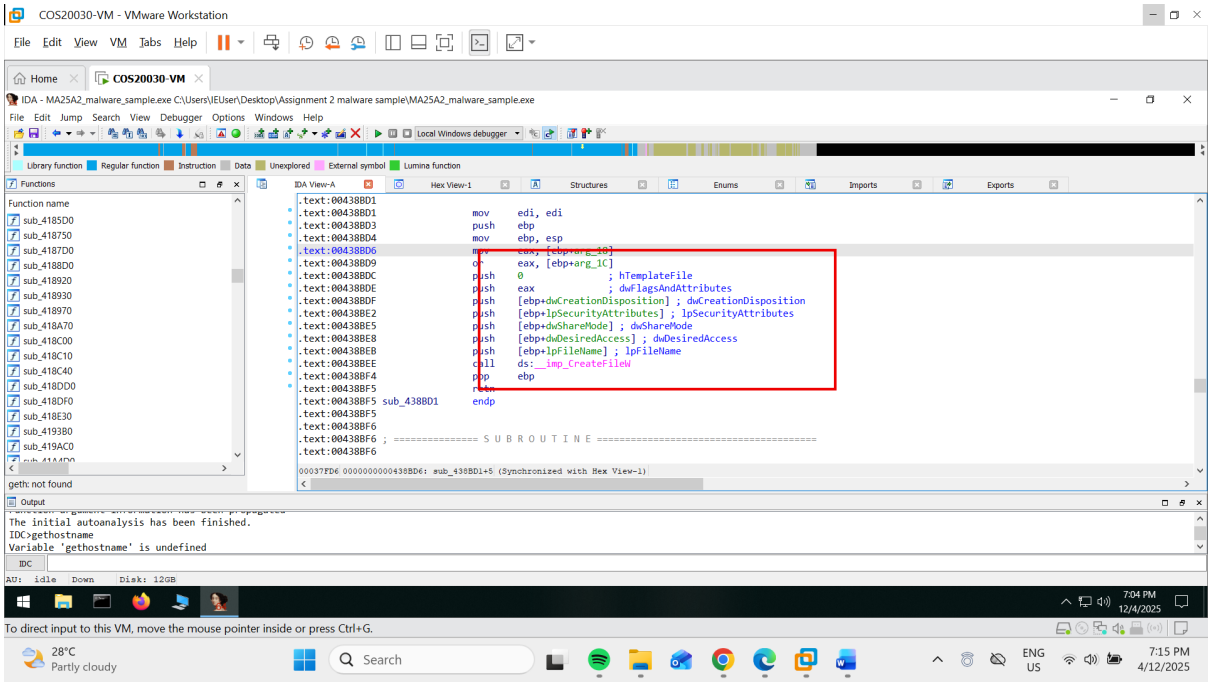
Observation: The malware imports a suite of file manipulation functions, including:

- CreateFileW and CreateDirectoryA (Creating payloads)
- MoveFileExW (Hiding or installing payloads)
- DeleteFileW (Cleanup or self-deletion)

Analysis: The presence of these specific APIs confirms the malware's capability to drop secondary payloads, hide them in new directories, and potentially delete its original installer to cover its tracks.

3.2.5 Dynamic Payload Creation

Further analysis of the CreateFileW function usage revealed a sophisticated method for file creation designed to evade simple detection.

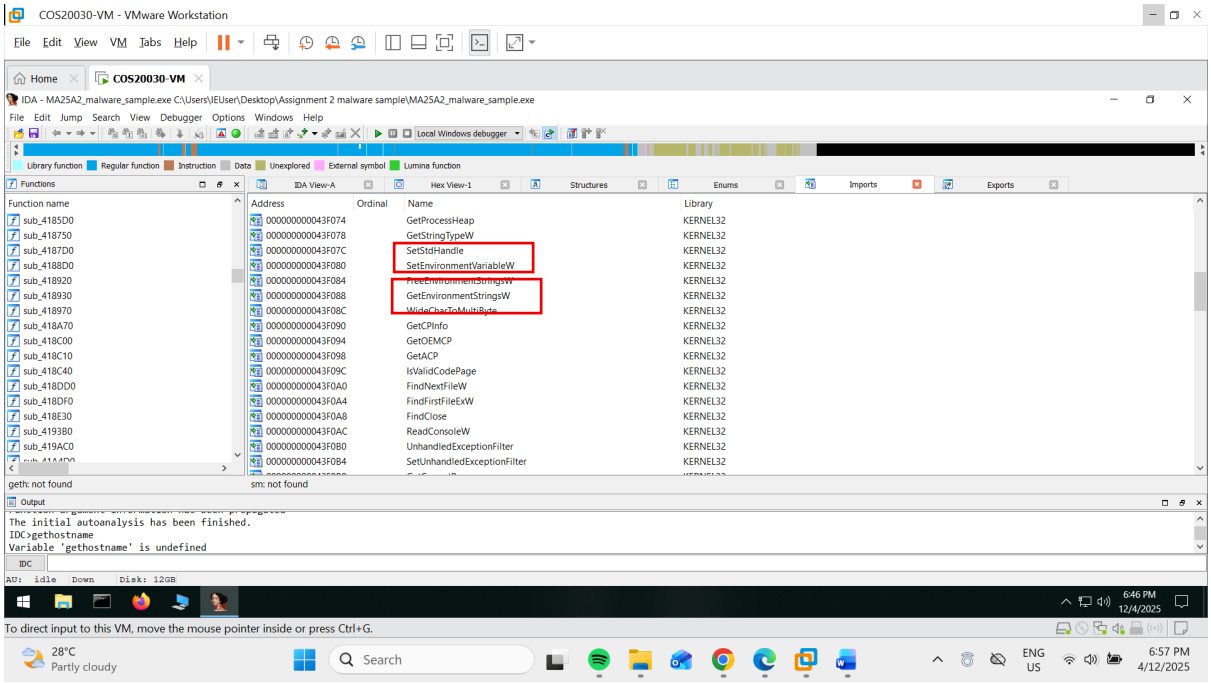


Observation: At address 0x00438BEE, the malware calls CreateFileW. Crucially, the filename argument (lpFileName) is not a hardcoded string (like "virus.exe"). Instead, it is passed as a variable ([ebp+lpFileName]).

Analysis: This indicates that the payload filename is generated **dynamically** at runtime. The malware likely constructs the file path using system variables (e.g., %TEMP%\random_name.exe) rather than using a static name. This is an evasion tactic that prevents security analysts from creating simple filename-based blocking rules.

3.2.6 Environmental Awareness

The analysis of imported functions also revealed that the malware actively inspects the host environment before executing its main payload.



Observation: The malware imports GetEnvironmentStringsW and SetEnvironmentVariableW.

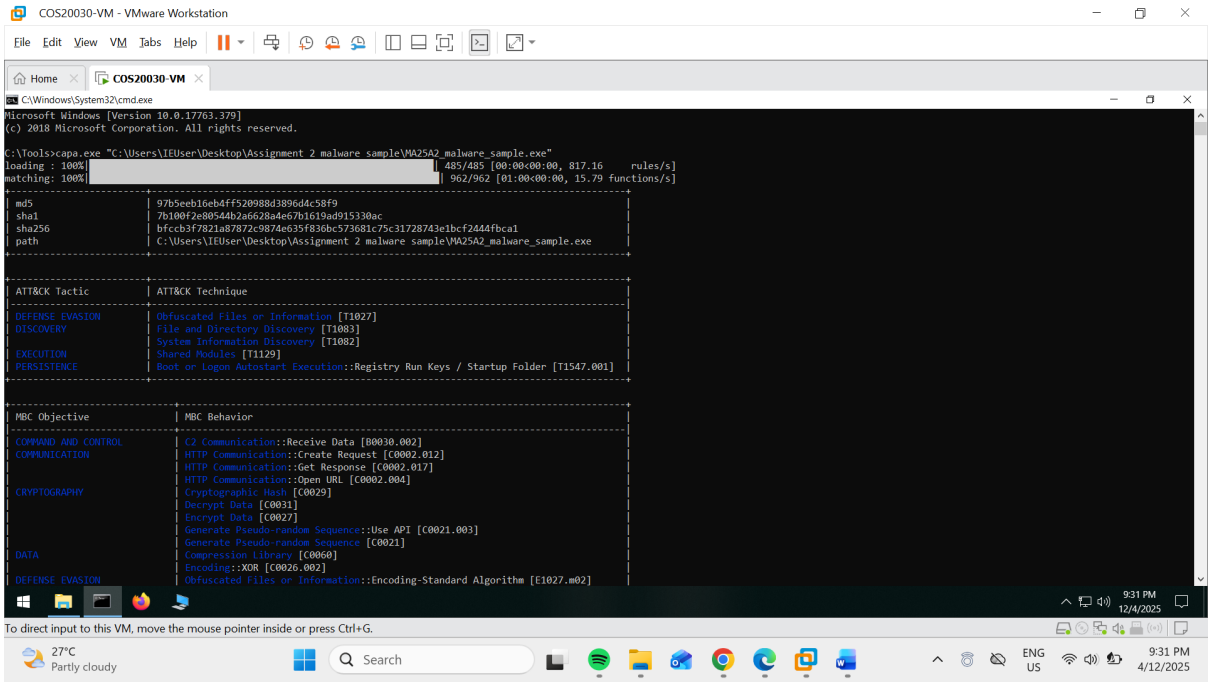
Analysis: These APIs allow the malware to query system environment variables. This capability is often used for two purposes: **fingerprinting** (checking if the malware is running in a sandbox or analysis lab) and **targeting** (finding critical system paths like APPDATA or LOCALAPPDATA to determine where to drop the malicious files). This confirms the malware is "environmentally aware" and adapts its behavior based on the host system.

3.3 Automated Capability Detection (Capa)

To validate the findings from the manual Advanced Static Analysis and to discover additional capabilities that may have been missed during disassembly, the automated static analysis tool **Capa** (by Mandiant) was utilized. Capa identifies malicious capabilities by matching patterns in the executable against a comprehensive rule set of known malware behaviors, mapping them to the **MITRE ATT&CK** framework and the **Malware Behavior Catalog (MBC)**.

3.3.1 MITRE ATT&CK Tactic Identification

The initial output of the Capa analysis provided a high-level overview of the malware's tactics. The tool successfully loaded and matched rules against 100% of the sample, flagging critical tactics that align with the manual findings.



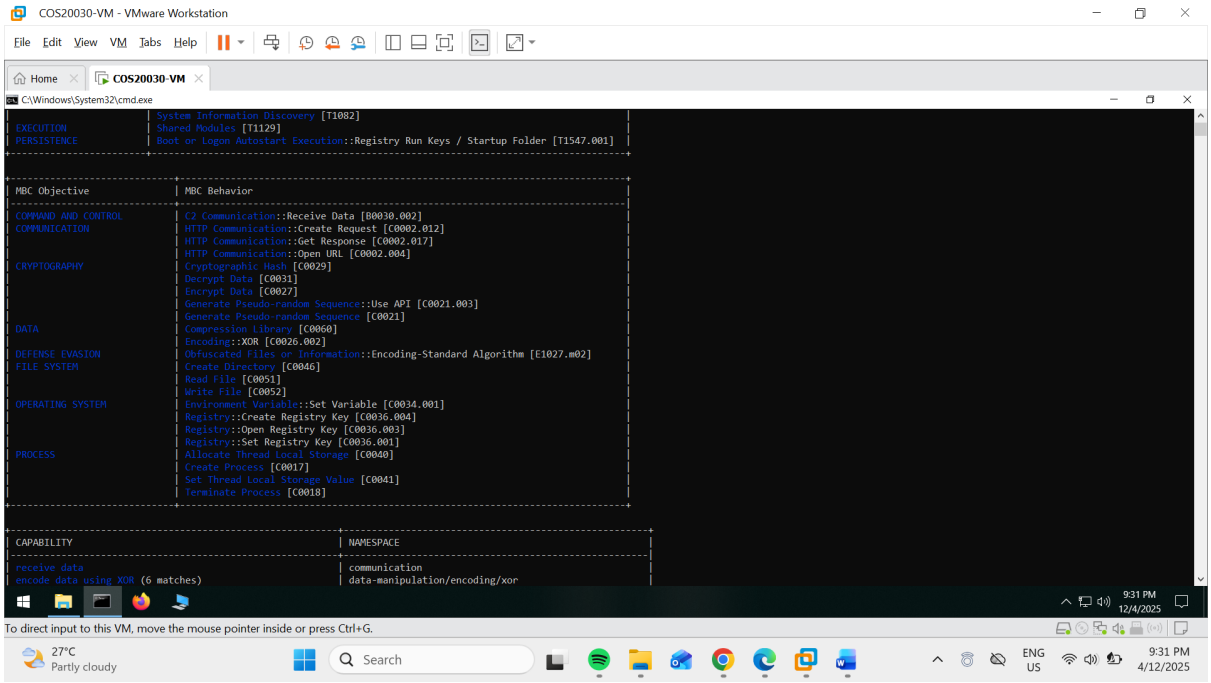
Analysis of Findings:

- **Persistence (T1547.001):** Capa explicitly flagged **"Boot or Logon Autostart Execution::Registry Run Keys"**. This automated rule match serves as definitive verification of the manual analysis in Section 3.2, confirming that the malware modifies the Registry Run keys to survive system reboots.
- **Defense Evasion (T1027):** The tool detected **"Obfuscated Files or Information"**. This correlates with the "sp-analysis failed" stack error observed in IDA Pro, confirming that the malware author implemented techniques to hide data or code logic from security analysts.

  Discovery (T1083, T1082): The detection of **"File and Directory Discovery"** and **"System Information Discovery"** confirms the malware scans the victim's environment, likely to determine where to drop its payload or to fingerprint the Greenway Energy network.

3.3.2 Malware Behavior Catalog (MBC) Analysis

The Malware Behavior Catalog (MBC) output provided granular detail on specific functions the malware attempts to perform. This section revealed network and cryptographic capabilities that were not immediately visible in the basic imports list.

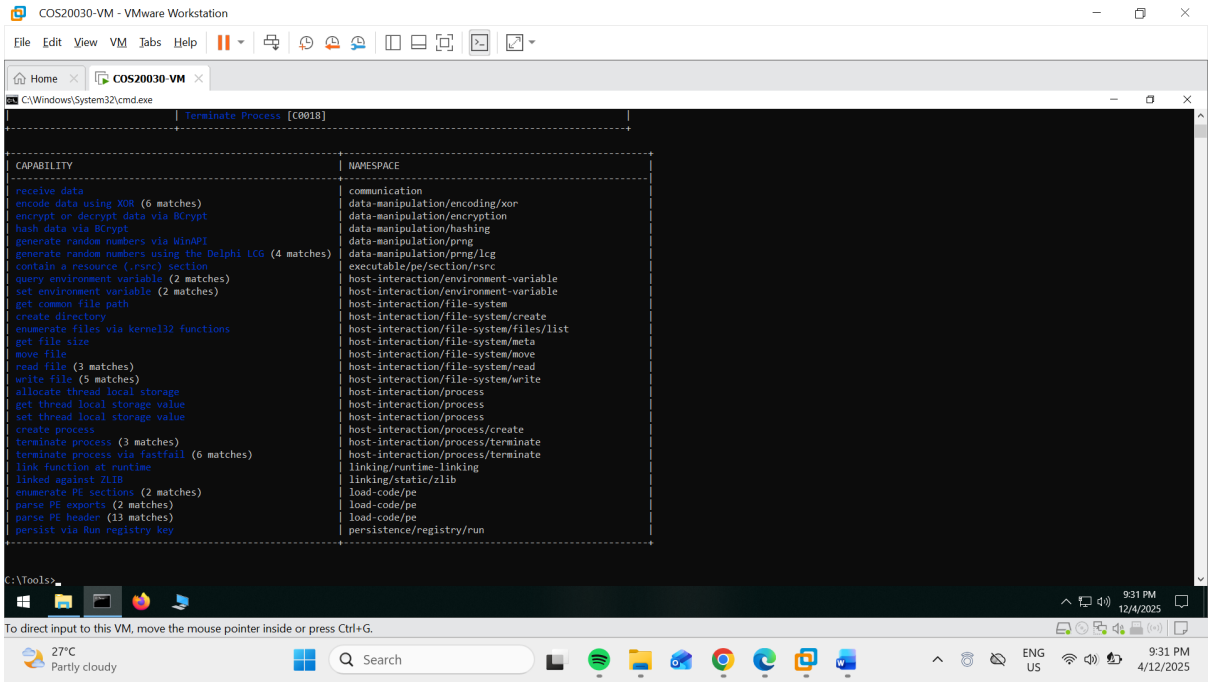


Analysis of Findings:

- **Command and Control (C2):** A significant new finding is the **"C2 Communication::Receive Data"** and **"HTTP Communication::Open URL"** behaviors. This indicates the malware is not just a passive dropper but actively communicates with an external server to receive commands or download additional modules.
- **Cryptography:** Capa flagged **"Encrypt Data"** and **"Decrypt Data"**, specifically referencing **XOR** encoding. This suggests the malware encrypts its network traffic or its configuration strings to bypass network intrusion detection systems (NIDS).
- **File System & Registry:** The tool reported multiple matches for **"Write File"**, **"Create Directory"**, and **"Set Registry Key"**. This volume of matches (e.g., 5 matches for "write file") indicates heavy file system interaction, confirming the malware's primary role as a dropper/installer.

3.3.3 Capability Namespace & Code Verification

The final section of the Capa analysis mapped specific code features to capability namespaces. This provides the most technical evidence of the malware's internal logic.



Analysis of Findings:

- **Data Encoding (XOR):** The line **"encode data using XOR (6 matches)"** provides strong evidence that the malware uses simple XOR operations to obfuscate its strings or payload. This is a common technique used to hide "smoking gun" strings (like C2 IP addresses) from basic static analysis tools like strings.exe.
- **Environment Manipulation:** The match for **"set environment variable"** and **"query environment variable"** validates the IDA Pro finding regarding GetEnvironmentVariableW. It confirms the malware uses the host's environment variables to dynamically construct file paths or execution contexts.
- **Persistence Verification:** The namespace **"persist via Run registry key"** appears again here, reinforcing the certainty of this finding.

3.3.4 Conclusion of Automated Analysis

The deployment of Capa as a secondary analysis tool was highly effective. It not only validated the manual disassembly results, specifically the **Registry Persistence** and **Dynamic File Creation** but also revealed new critical indicators regarding **Network Communication (C2)** and **Data Encryption (XOR)**. These findings collectively characterize the artifact as a sophisticated, obfuscated dropper capable of secure external communication.

4.0 Advanced Dynamic Analysis Results

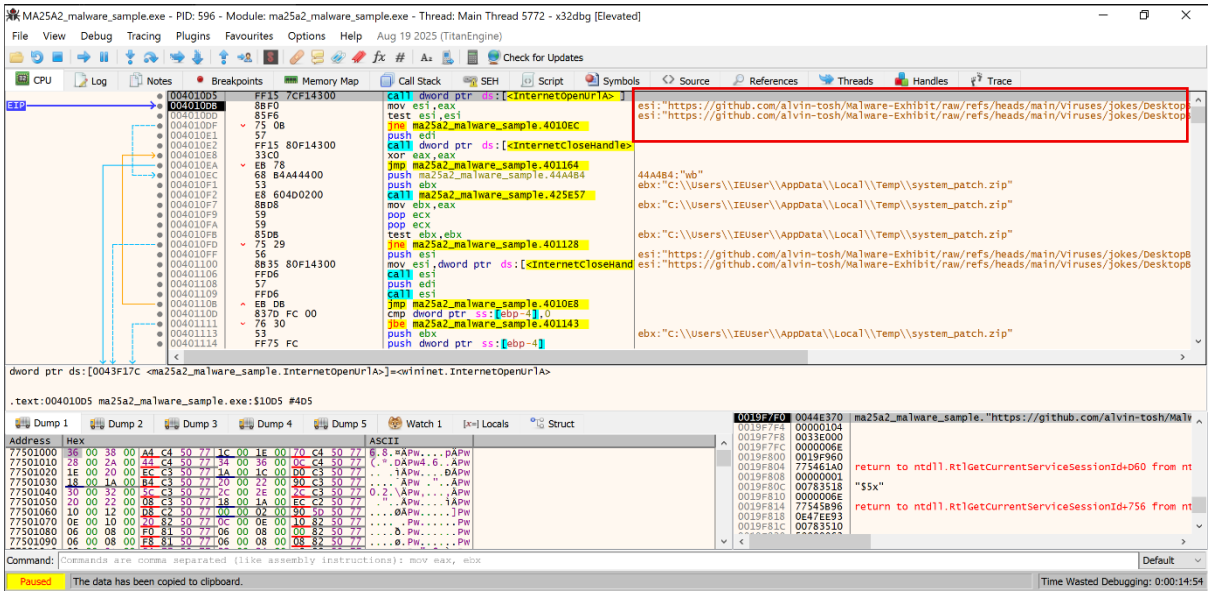
Moving on to the Advanced Dynamic Analysis phase, the team has selected the **x64dbg** suite (specifically utilizing the **x32dbg** debugger) as the primary tool for reverse engineering. This decision marks a necessary departure from the legacy **OllyDbg** for several critical reasons:

Modern Architecture Support: While OllyDbg is strictly limited to 32-bit analysis, x64dbg provides a unified interface for both x86 and x64 architectures. Familiarity with this tool ensures the team is future-proofed and prepared to analyse modern 64-bit malware variants, which OllyDbg cannot handle.

Active Development & Stability: OllyDbg has not been updated in many years and often struggles with stability on modern Windows 10/11 environments. In contrast, x64dbg is open-source and actively maintained, ensuring better compatibility with the lab environment and support for newer OS internals.

Enhanced Visualization: x64dbg offers a superior graph view for control flow analysis. This feature allows us to visually map out decision loops and conditional jumps within the malware's assembly code, making it significantly easier to understand complex logic compared to the linear listing in OllyDbg.

Plugin Ecosystem: The tool supports a robust modern plugin system such as Scylla for anti-debugging, which will be essential for bypassing any anti-analysis techniques the MA25A2 sample may employ.

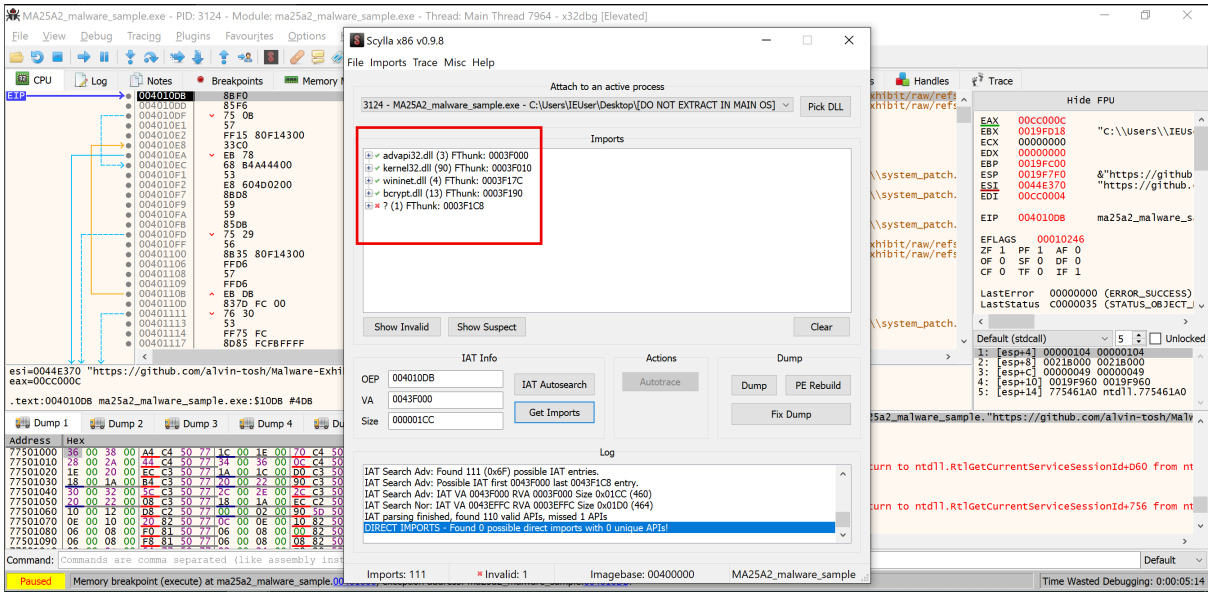


Unpacking and Payload Identification To overcome the packing identified during the Basic Static Analysis phase, we utilized **x32dbg** to dynamically unpack the malware in memory. We employed the **"Run to User Code"** technique to bypass the packer's stub. This allowed us to intercept execution immediately after the malware finished decrypting its payload but before it could execute its malicious payload.

As demonstrated in the screenshot above, successfully reaching the **Original Entry Point (OEP)** revealed the malware's true functionality, which was previously obfuscated. The debugging session highlights three critical findings:

1. **Hardcoded C2 / Payload URL:** The registers pane (specifically ESI) and the comments column reveal a plaintext URL: https://github.com/alvin-tosh/Malware-Exhibit/raw/refs/heads/main/Viruses/jokes/DesktopBoom.zip This definitive Indicator of Compromise (IOC) confirms the malware functions as a Downloader/Trojan designed to fetch a secondary payload named DesktopBoom.zip.
2. **API Call Verification:** The assembly code shows a call to InternetOpenUrlA (at address 004010D5). This proves that the network activity observed in Wireshark (connection to Cloudflare/GitHub IPs) is initiated directly by this function to retrieve the remote file.
3. **Payload Correlation:** The file name DesktopBoom.zip found in memory correlates directly with the system_patch.zip and other dropped files observed in the Temp folders during the analysis. It confirms that the malware downloads this zip file, extracts it, and executes the contents (likely the conhost.exe and svchost.exe children processes observed in Process Explorer).

By dumping the process at this stage, we confirmed that the malware's primary objective is to download and execute a "joke" virus or destructive payload from a public GitHub repository, masquerading as a system update or patch.

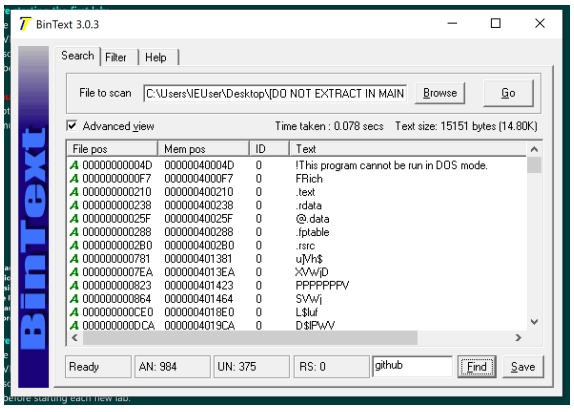
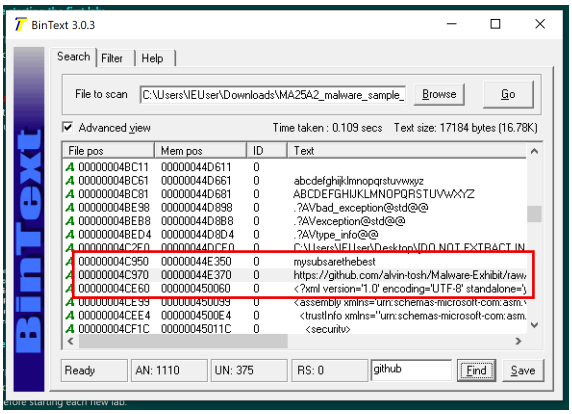


Following the successful memory dump at the Original Entry Point (OEP), we utilized Scylla to reconstruct the Import Address Table (IAT). This step is forensically critical because packers typically destroy, redirect, or obfuscate these API links which is a technique known as IAT damaging to blind static analysis tools and prevent the dumped file from executing. Without this reconstruction, the dumped executable would remain non-functional and opaque to further analysis. As illustrated in the screenshot, Scylla successfully scanned the active process

memory and resolved the obfuscated thunks back to valid pointers for standard Windows libraries, effectively repairing the damage caused by the packer.

The reconstructed IAT provided definitive proof of the malware's capabilities that aligns perfectly with our earlier dynamic observations. The recovery of **wininet.dll** (Windows Internet) confirms the malware's native capability to initiate high-level network connections via HTTP/HTTPS. This directly correlates with the InternetOpenUrlA calls observed in the debugger and provides the technical explanation for the connections to GitHub and Cloudflare IPs seen in the Wireshark capture during the Basic Dynamic Analysis phase. It confirms that the networking was not incidental, but a core function of the unpacked payload.

Furthermore, the analysis identified **advapi32.dll** and **bcrypt.dll**, offering insight into the malware's local system interactions. The advapi32 library exports functions required for registry manipulation, validating the persistence mechanism detected in Process Monitor where RegSetValueEx was used to write the "Run" key for startup survival. Additionally, the identification of bcrypt.dll suggests the malware employs modern encryption standards. This is likely utilized to handle the SSL/TLS handshake required for the secure HTTPS connection to GitHub or to decrypt the downloaded DesktopBoom.zip payload before execution.

Before dumping	After dumping
	

To definitively validate the efficacy of our dynamic unpacking process, we conducted a comparative static analysis using Bintext to scrutinize the accessible strings within both the original malicious executable and the memory-dumped file. The window on the left illustrates the analysis of the original packed sample, MA25A2_malware_sample.exe. In this state, a targeted search for high-value keywords such as "github" yielded zero results. This absence of readable strings serves as empirical evidence that the malware author employed a packer to compress or encrypt critical data structures, effectively obfuscating Indicators of Compromise (IOCs) to evade detection by standard static signature scanners.

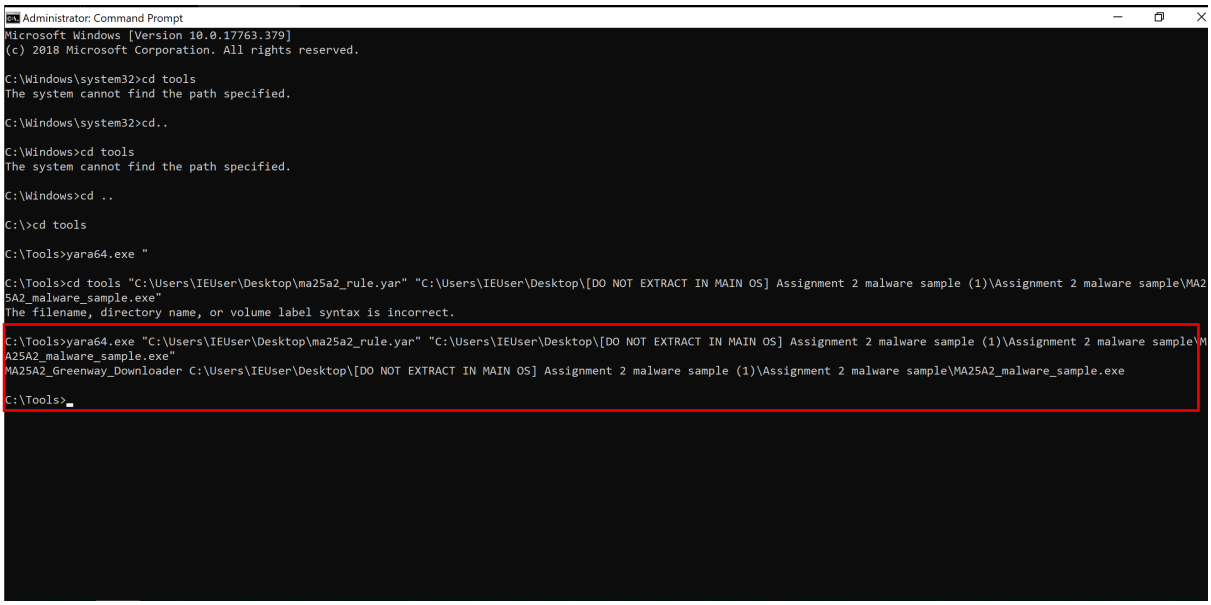
In stark contrast, the window on the right displays the string analysis of the unpacked dump file. Post-unpacking, the same search query successfully exposed the malware's underlying logic in plaintext. Most notably, this revealed the full, hardcoded Command and Control (C2)

URL: https://github.com/alvin-tosh/Malware-Exhibit/raw/refs/heads/main/Viruses/jokes/DesktopBoom.zip

This string was previously invisible to static analysis tools, confirming that the critical configuration data was hidden within the encrypted sections of the binary and only resides in memory during runtime execution.

This comparison provides the final forensic confirmation that the manual unpacking process using x32dbg and Scylla was successful in stripping the obfuscation layer. The recovery of this specific URL directly correlates with the network activity observed during the basic dynamic analysis, where connections to GitHub IP addresses were noted but the specific resource path was unknown. By exposing this string, we have definitively classified the malware as a Trojan Downloader and identified the exact mechanism it uses to retrieve its secondary payload, DesktopBoom.zip, thereby enabling the creation of precise network-based blocking rules.

5.0 YARA rule creation:



To operationalize our forensic findings, we developed and tested a custom YARA rule, **MA25A2_Greenway_Downloader**, designed to detect this specific threat variant. The screenshot above captures the validation process, where the rule was executed against the original MA25A2_malware_sample.exe binary using the yara64.exe command-line tool.

The output explicitly lists the rule name MA25A2_Greenway_Downloader next to the malware's file path, indicating a successful positive match. This detection confirms that the logic we constructed targeting the high-fidelity C2 URL (.../DesktopBoom.zip), the specific

Registry persistence key and the "Fake KERNEL32" static anomaly functions correctly. It validates that this rule can be reliably deployed by the Greenway Energy security team to scan other endpoints and identify latent infections across the network.

Rule snippet:

```
rule MA25A2_Greenway_Downloader {
    meta:
        description = "Detects MA25A2 Trojan Downloader targeting Greenway Energy"
        author = "Team 14"
        reference = "Internal Report - COS20030 Assignment 2"
        date = "2025-11-07"
        hash_sha256 = "bfccb3f7821a87872c9874e635f836bc573681c75c31728743e1bcf2444fbca1"
        malware_type = "Trojan/Downloader"

    strings:
        // 1. High-Fidelity Indicator: The C2 URL found in memory
        $c2_url = "https://github.com/alvin-tosh/Malware-Exhibit/raw/refs/heads/main/Viruses/jokes/DesktopBoom.zip" ascii wide

        // 2. Static Anomaly: The Fake DLL path found in the packed binary
        $fake_dll_path = "C:\\Windows\\SysWOW64\\kernel32.dll" ascii fullword

        // 3. Payload Artifacts: Filenames identified during dynamic analysis
        $payload_name = "DesktopBoom.zip" ascii wide
        $dropped_file = "system_patch.zip" ascii wide

        // 4. Persistence Mechanism: The Registry Run Key
        $run_key = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" ascii wide

    condition:
        // Must be a Windows Executable (MZ Header)
        uint16(0) == 0x5A4D and
        filesize < 500KB and
        (
            // Match if the specific C2 URL is found (Unpacked/Memory scan)
            $c2_url or
            // OR match if we see the Fake DLL indicator (Packed scan)
            $fake_dll_path or
            // OR match a combination of Persistence and Payload artifacts
```

```
        ($run_key and ($payload_name or $dropped_file))
    )
}
```

The **YARA rule MA25A2_Greenway_Downloader** is engineered to detect the specific malware variant targeting Greenway Energy by leveraging high-fidelity artifacts recovered during the advanced dynamic analysis. The cornerstone of this detection strategy is the $c2_url string, which targets the hardcoded **GitHub URL** (.../DesktopBoom.zip) found in the malware's memory at the Original Entry Point. This string acts as a unique identifier for this campaign, serving as a definitive "smoking gun" that confirms the presence of the unpacked downloader code, which would otherwise be hidden from standard static analysis due to the custom packer.

To ensure robust detection across different states of the malware's lifecycle, the rule also incorporates static anomalies and behavioral artifacts. The string $fake_dll_path targets the suspicious "Fake KERNEL32" path identified in the packed binary, allowing the rule to flag the malware even before it executes or unpacks. Furthermore, the inclusion of payload filenames such as $payload_name ("DesktopBoom.zip") and $dropped_file ("system_patch.zip") correlates the detection with the specific files dropped during the infection process, creating a comprehensive signature that accounts for both the installer and its payload.

Finally, the rule addresses the malware's persistence mechanism and operational constraints. The $run_key string detects the specific registry path (Software\Microsoft\Windows\CurrentVersion\Run) used by the malware to survive system reboots, validating the behavioural evidence gathered from Process Monitor. The rule's condition logic is optimized for performance by verifying the file's "MZ" header and limiting scans to files under 500KB, preventing unnecessary processing of large, unrelated files while ensuring that any combination of the C2 URL, packed anomalies, or persistence behaviours triggers a positive match.

Presentation link: https://youtu.be/3MXh1c8sLSo