Reversing and Technical Analysis of Agent Tesla:

Introduction:

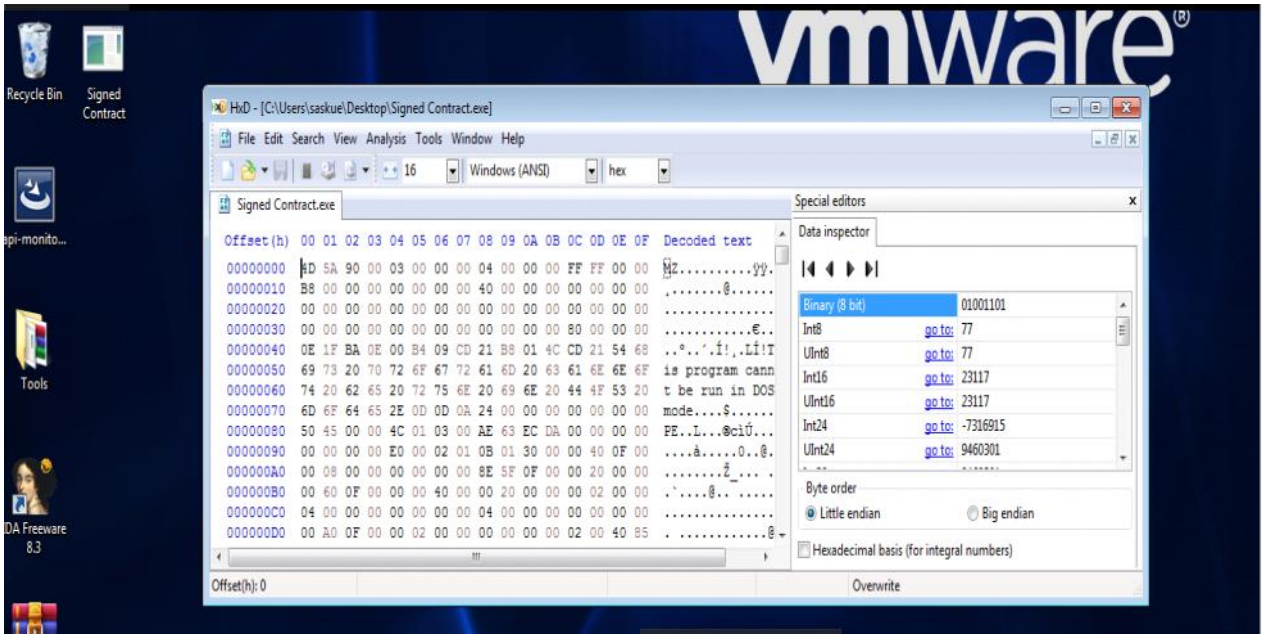
Agent tesla is a malware which is RAT with functionality dealing with stealing sensitive information from victim computer. It is written in .NET and collect information such as keystrokes and login credentials, just like my previous paper on reversing agniane stealer. It was first seen in wild year 2014.

Methodology:

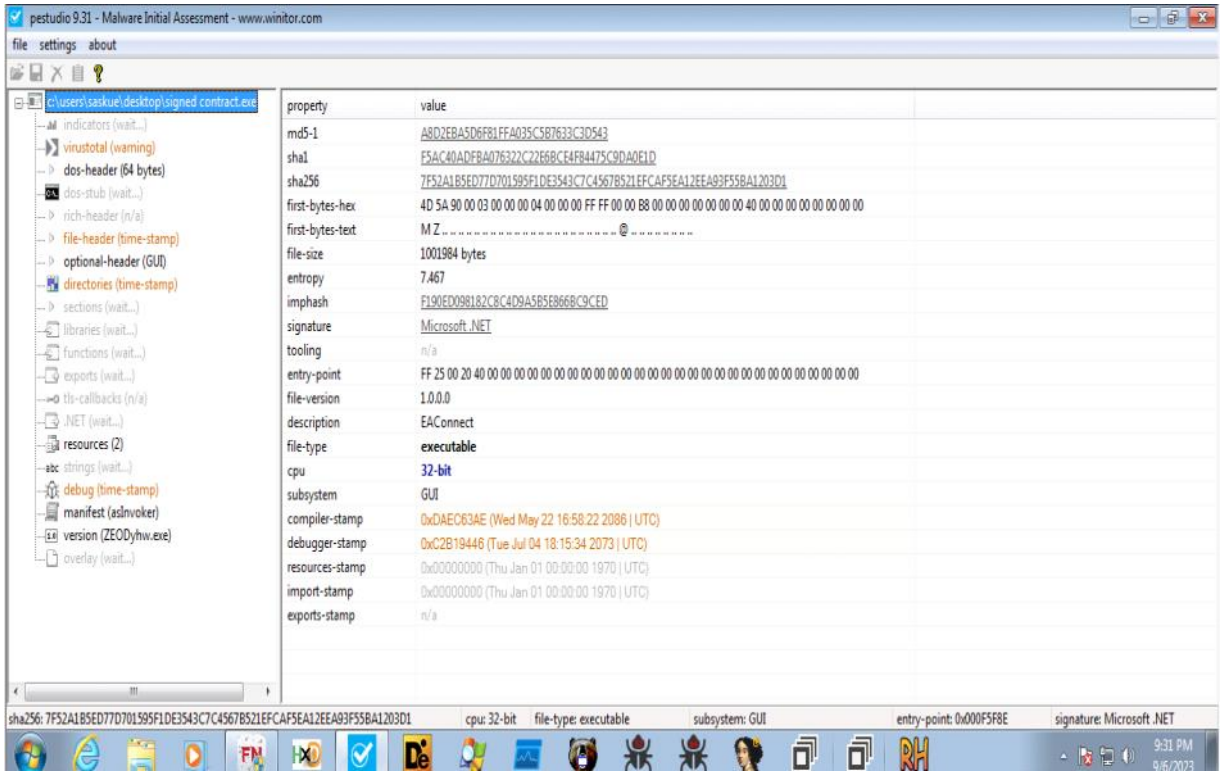
Initial Static Analysis
Initial Dynamic Analysis
Unpacking of the malware
Decryption of code
Reversing and technical analysis
Detection content
Yara rules
Sigma rules
Mapping to MITRE Framework

Initial Static Analysis:

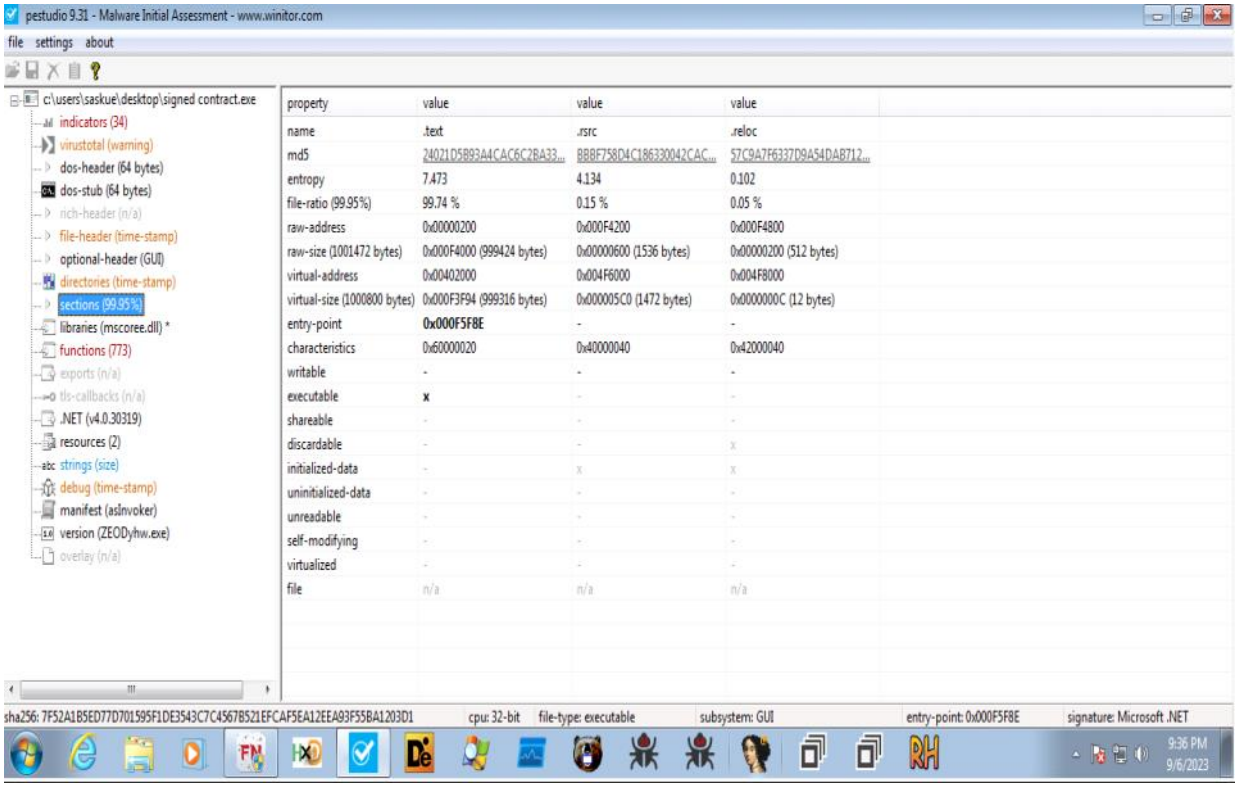
I will start with hex editor to see what kind of file are we dealing with, is it a standard PE file or non PE file. I can see that it's a standard PE file as show by hex editor. So let's move further with the analysis.



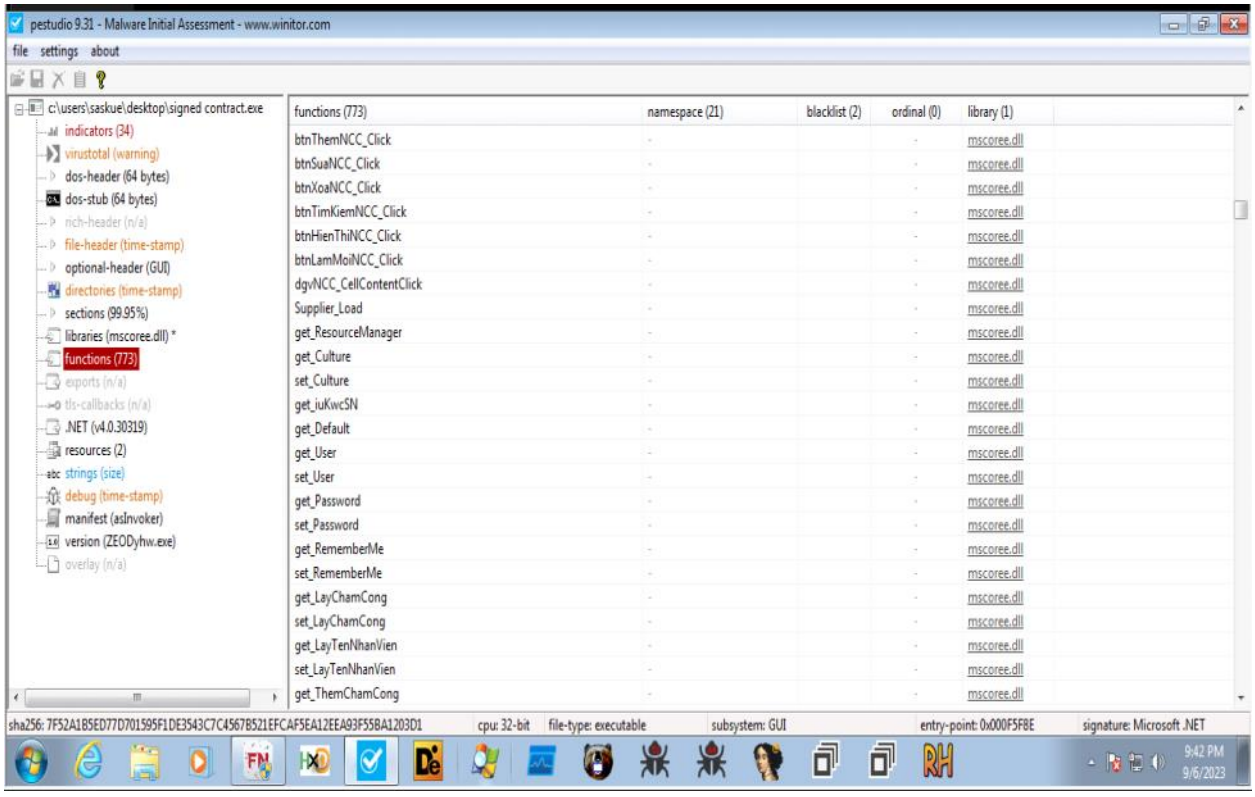
We will now move ahead with our initial static analysis and for this I am going to stick with PESTUDIO and DIE for our analysis. Pestudio reveals that this is a .NET sample and 32-bit. It seems that the sample is compiled on 22 May but then I don't rely on such information as it can be modify.



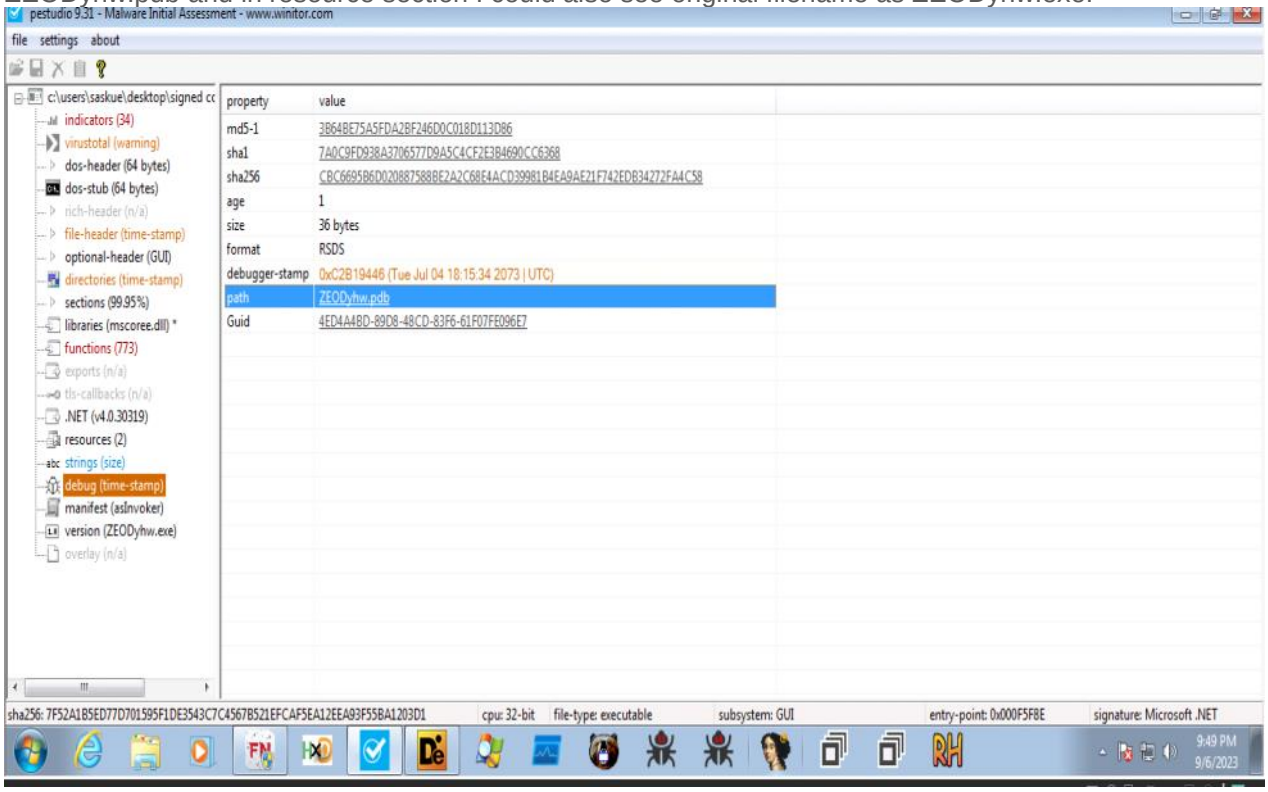
When I looked over the sections I see that the entropy for text section is high and there might be possibility that next payload could be mapped from this section.
Entropy :
Text section ==> 7.473 (Very High)

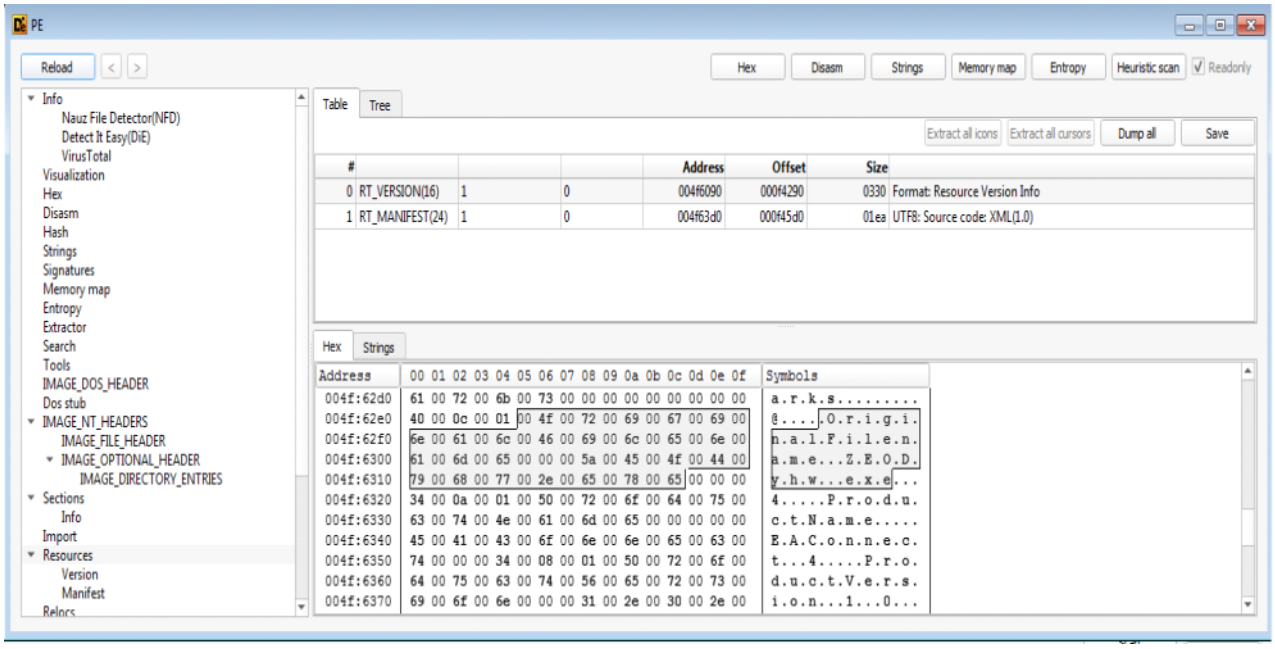


Let's have a look over functions, I could see an interesting string called ZEODyhw. Apart from that other interesting function name was get_password, set_password etc, seems like a password stealer too.
The function name resemble kind of asian ascent.

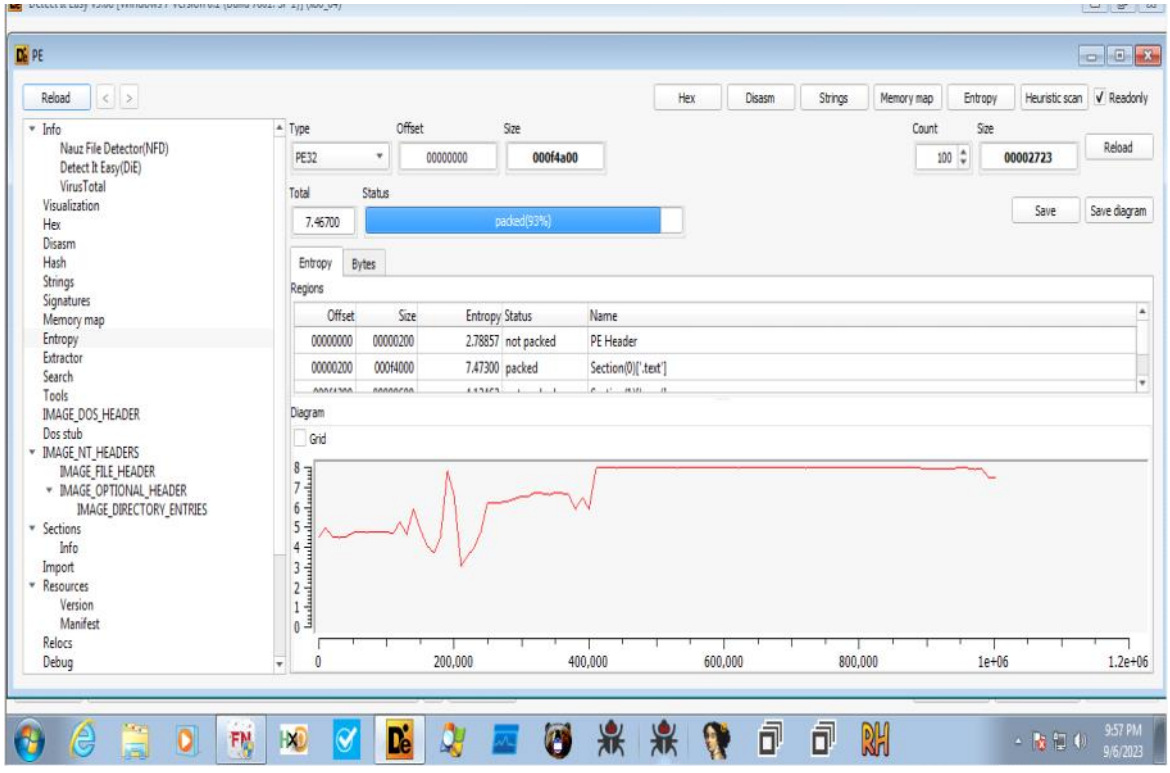


It seems the binary name is ZEODyhw.exe as I could also see in debugging directory PDB name as ZEODyhw.pdb and in resource section I could also see original filename as ZEODyhw.exe.



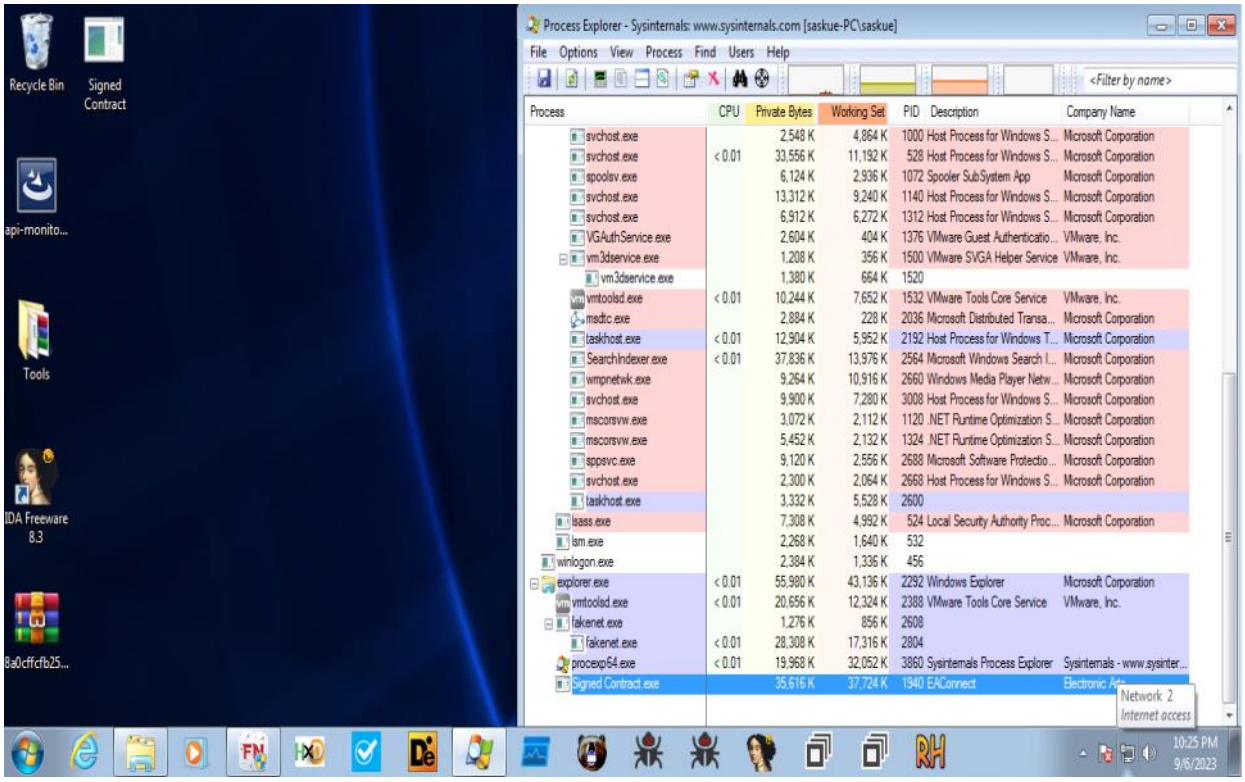


DIE detect it also as .NET executable and also considered it as packed too so there is not much to do except unpack the next payload for further analysis
Based on my assumption the initial vector from where the next payload could be mapped into memory is possibly the text section as starting point.

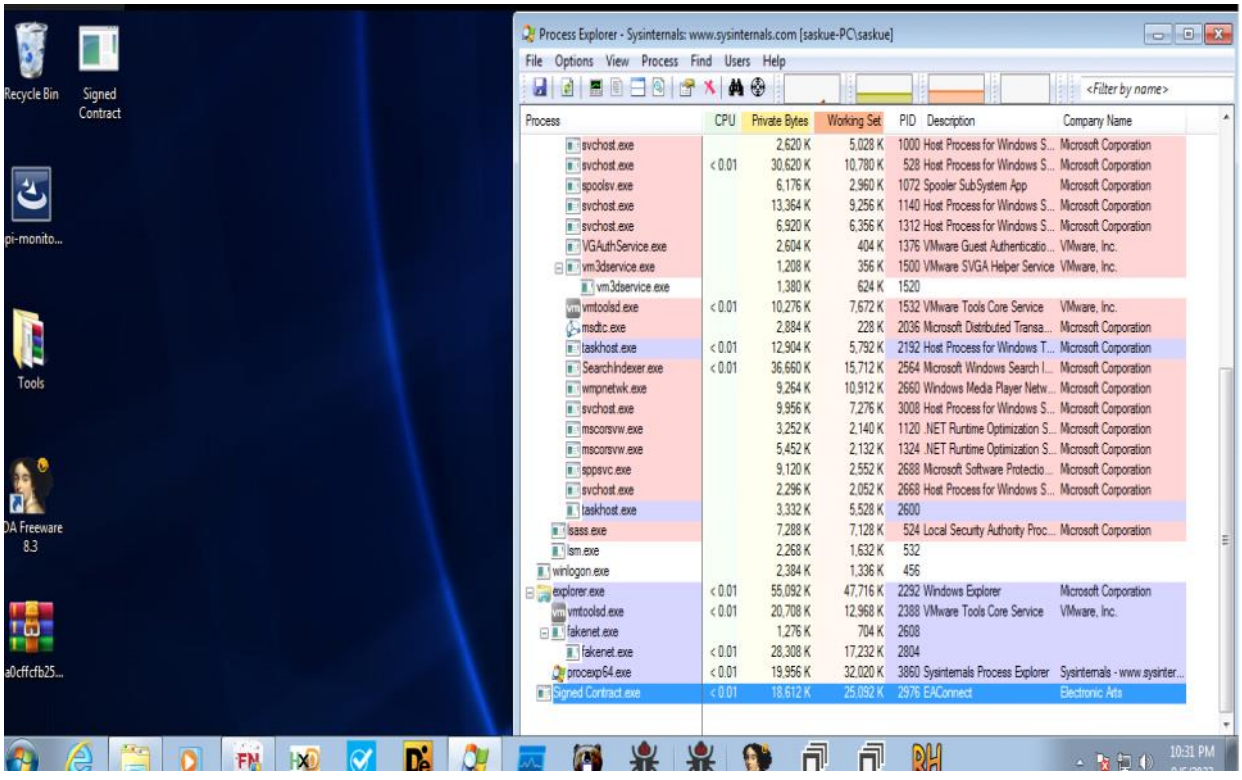


Initial Dynamic Analysis:

For the initial Dynamic Analysis I will stick to Process Explorer & ProcMon to observer the behavior of malware and how unpacking is done. Malware started with processid 1940 so let's keep observing the behavior of malware.



It seems to create a new process and then exit the old process, seems process injection has been performed.

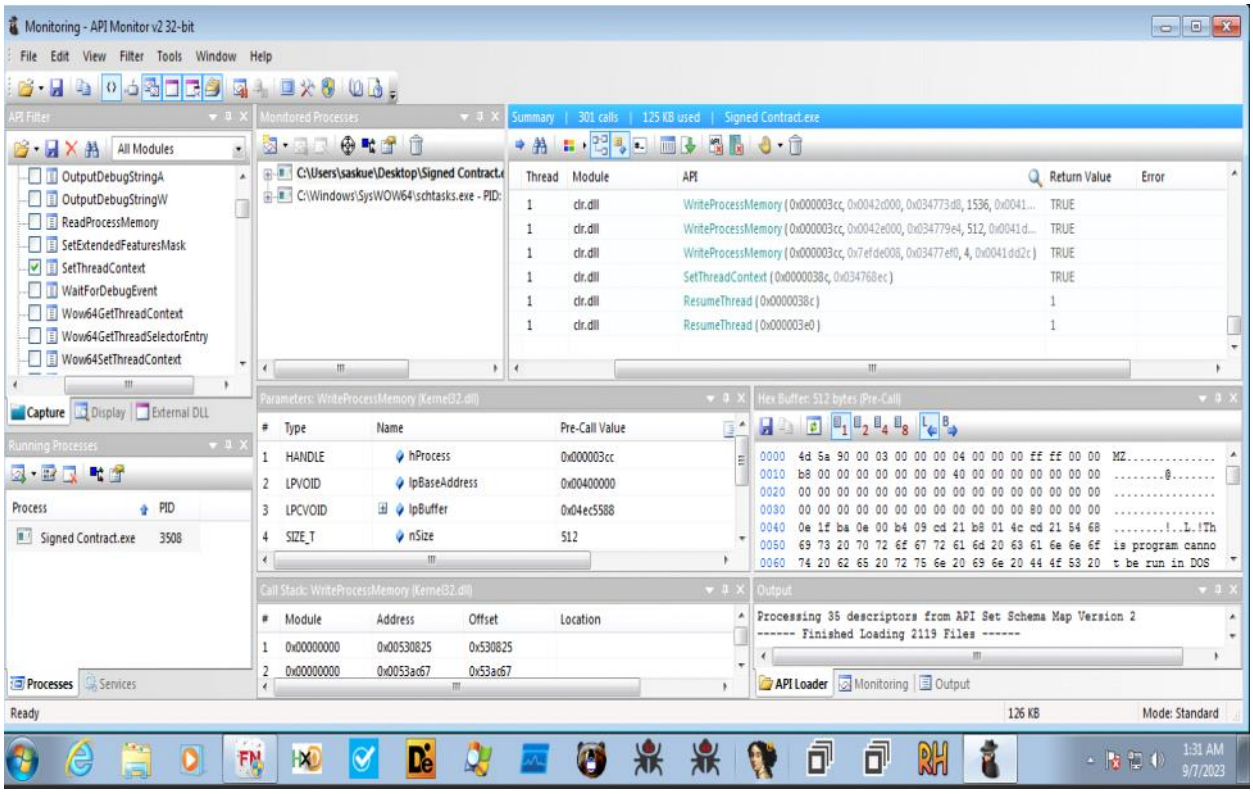


Now that we know the flow of unpacking here is how I am gonna unpacked the main malware and extract it, for this we are going to use x64dbg and will monitor the api :

1-VirtualAlloc
2-VirtualAllocEx
3-VirtualProtect
4- CreateProcessA/w
5-WriteProcessMemory

6-CreateRemoteThread
7-ResumeThread/SethreadContext

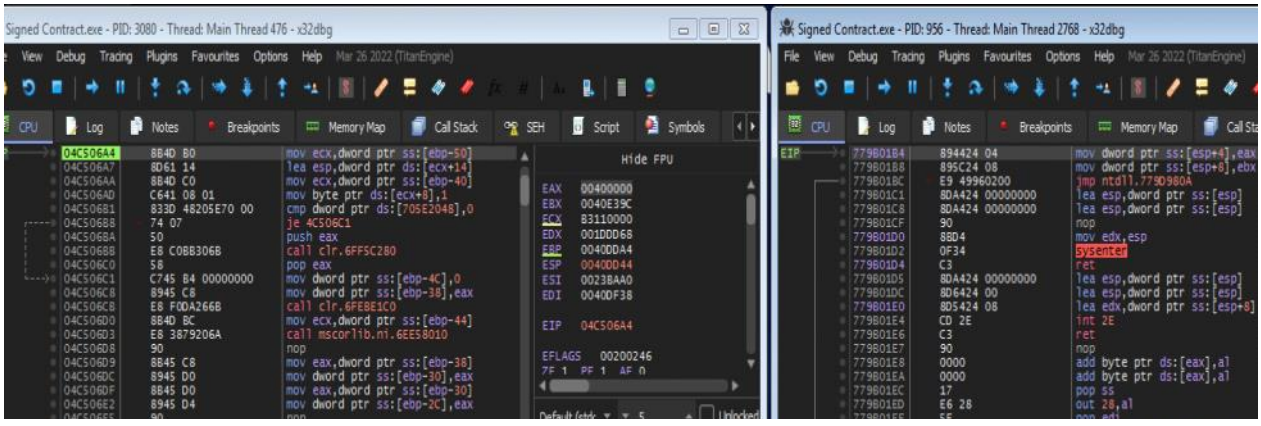
By monitoring these windows api I saw that the main payload in packed and written to remote remote process by these api in order:
1- VirtualAlloc
2- CreateProcessA
3- VirtualAllocEx
4- WriteProcessMemory
5- SetThreadContext
6- ResumeThread

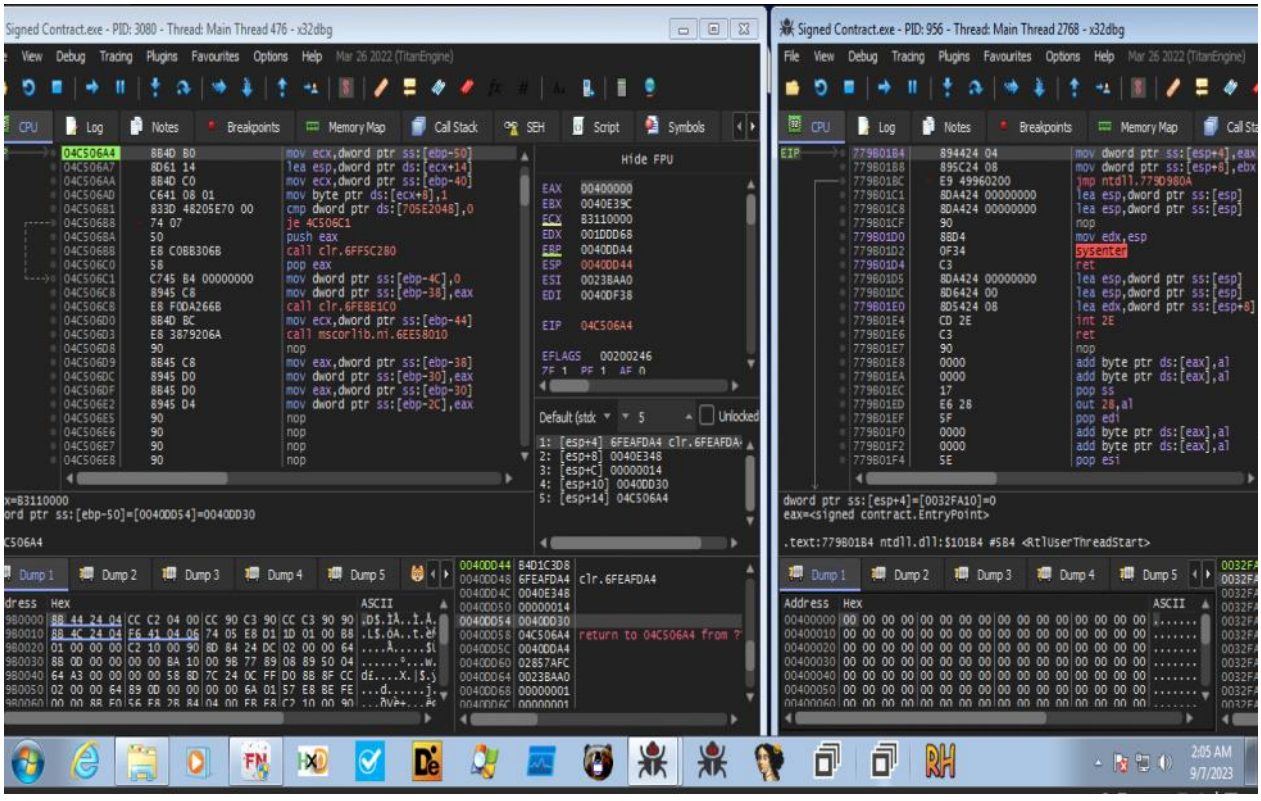


Unpacking of the malware:

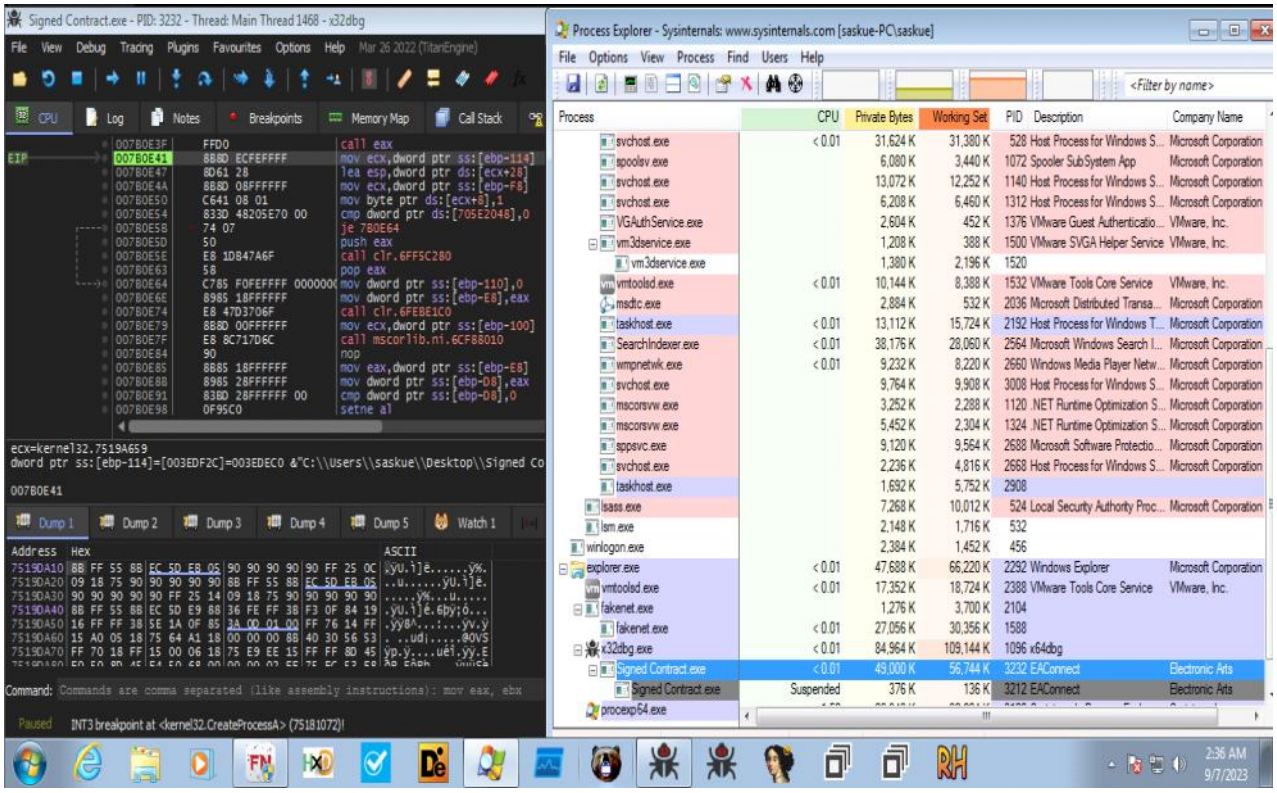
For the unpacking of the malware we are going to use x32dbg and will set breakpoint on appropriate api and will extract the payload once the whole malware is mapped into memory.

I have already put the breakpoint on api and also on newly created process and will wait for the malware to write into remote process and will then extract the agent tesla from the newly created process memory.

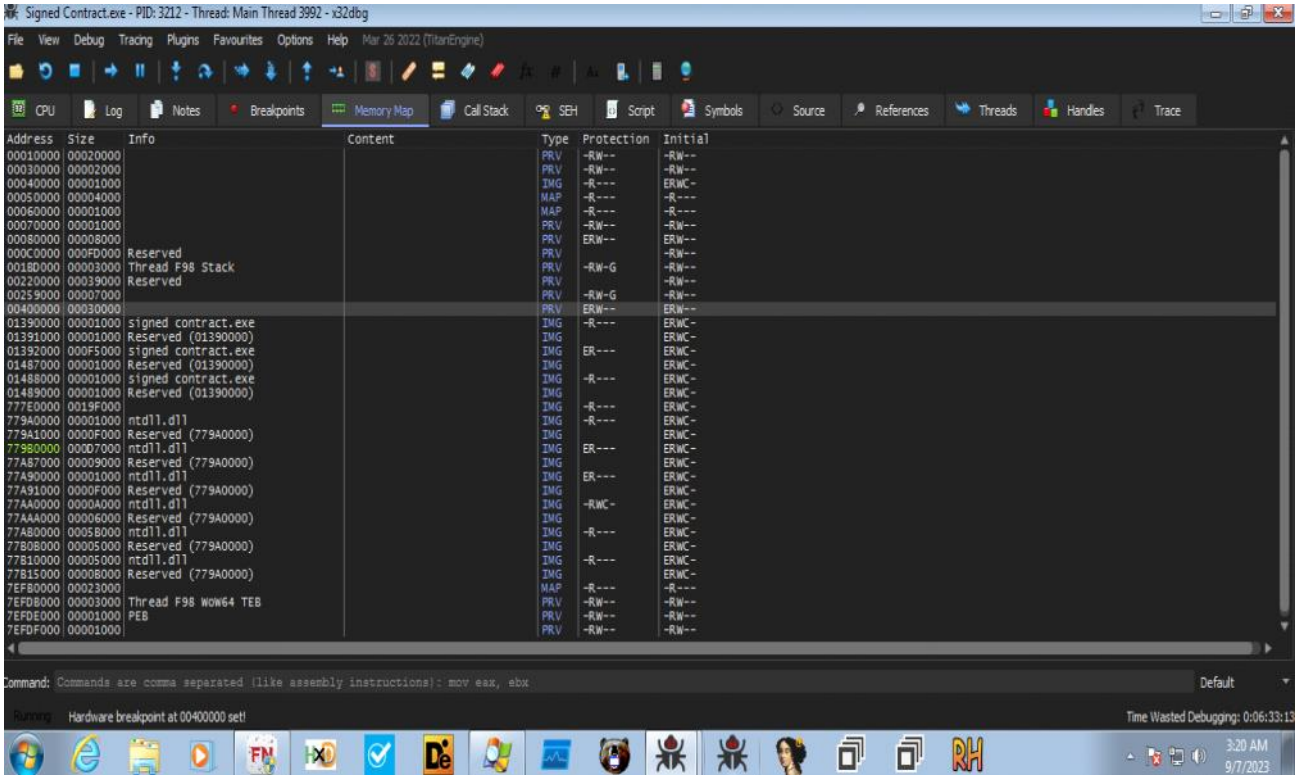




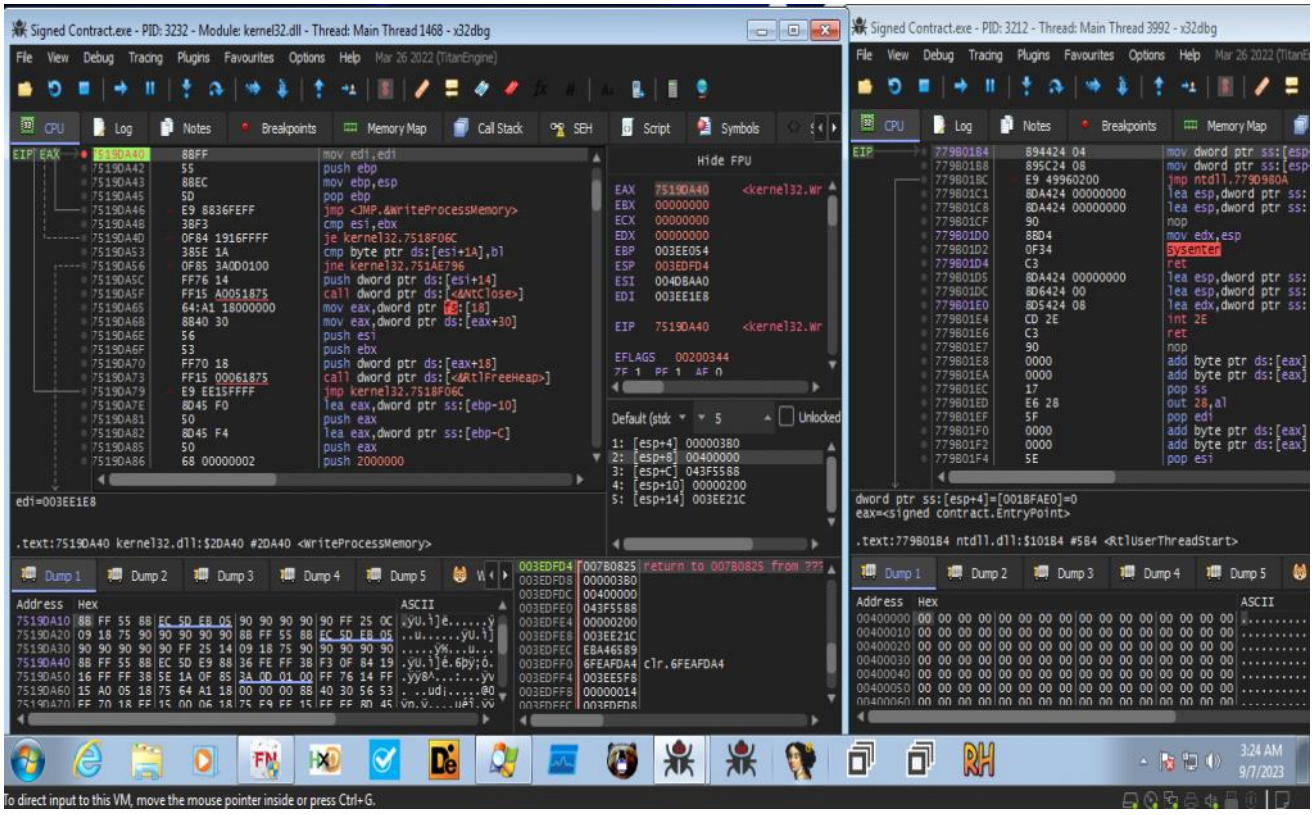
You can see that a new process is created in suspended state too.



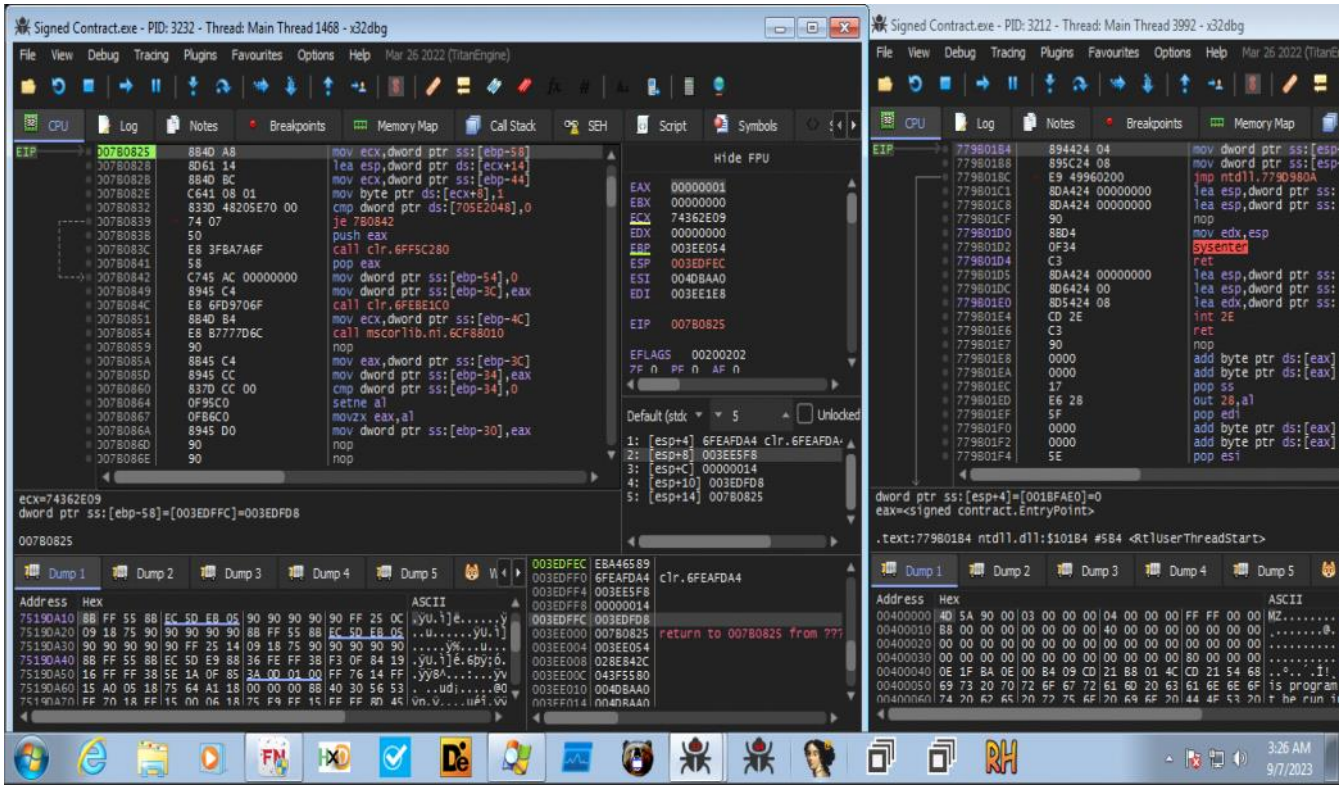
If we look at memory map of the newly created process we will observed that a new memory has been allocated with memory permission as ERW (Executable-Readable-writeable). Possibly this region could be used to inject the main malware.



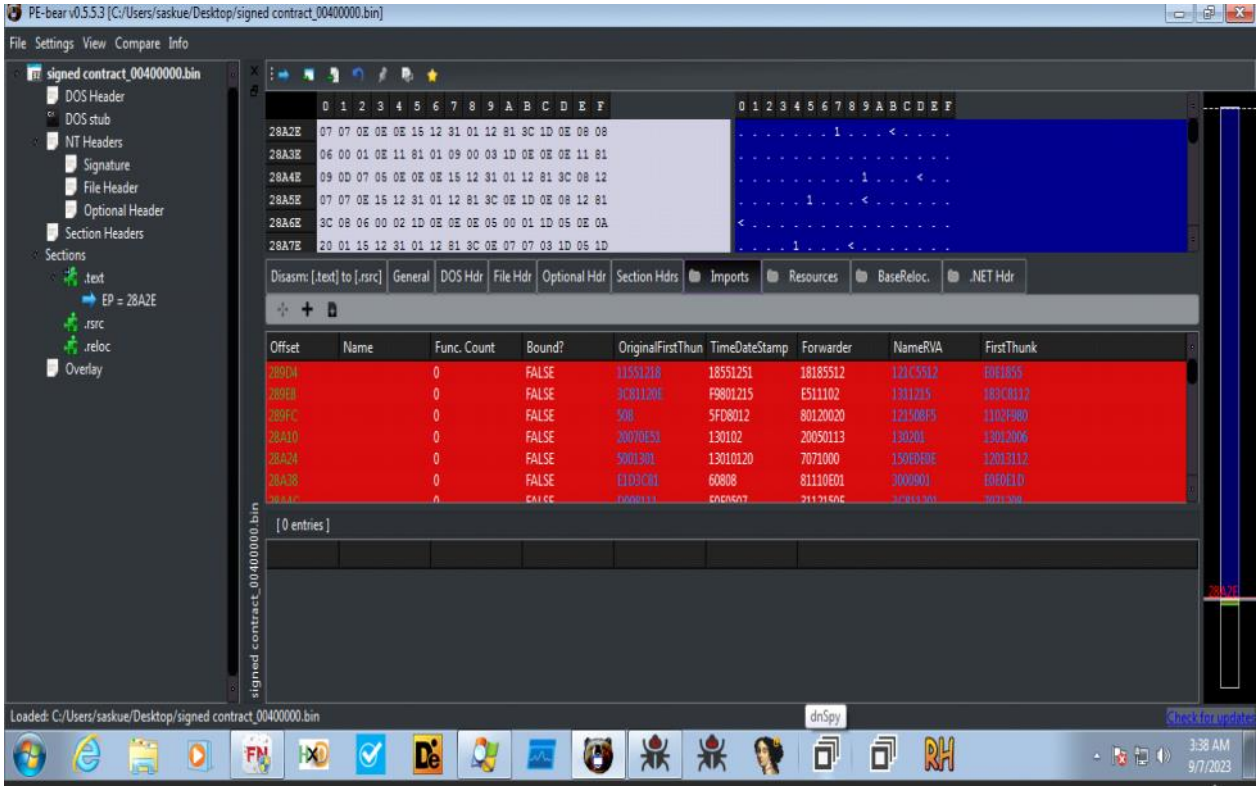
As soon as we hit the writeprocessmemory api we could see it being used to write in that same particular memory region for the newly created remote process.



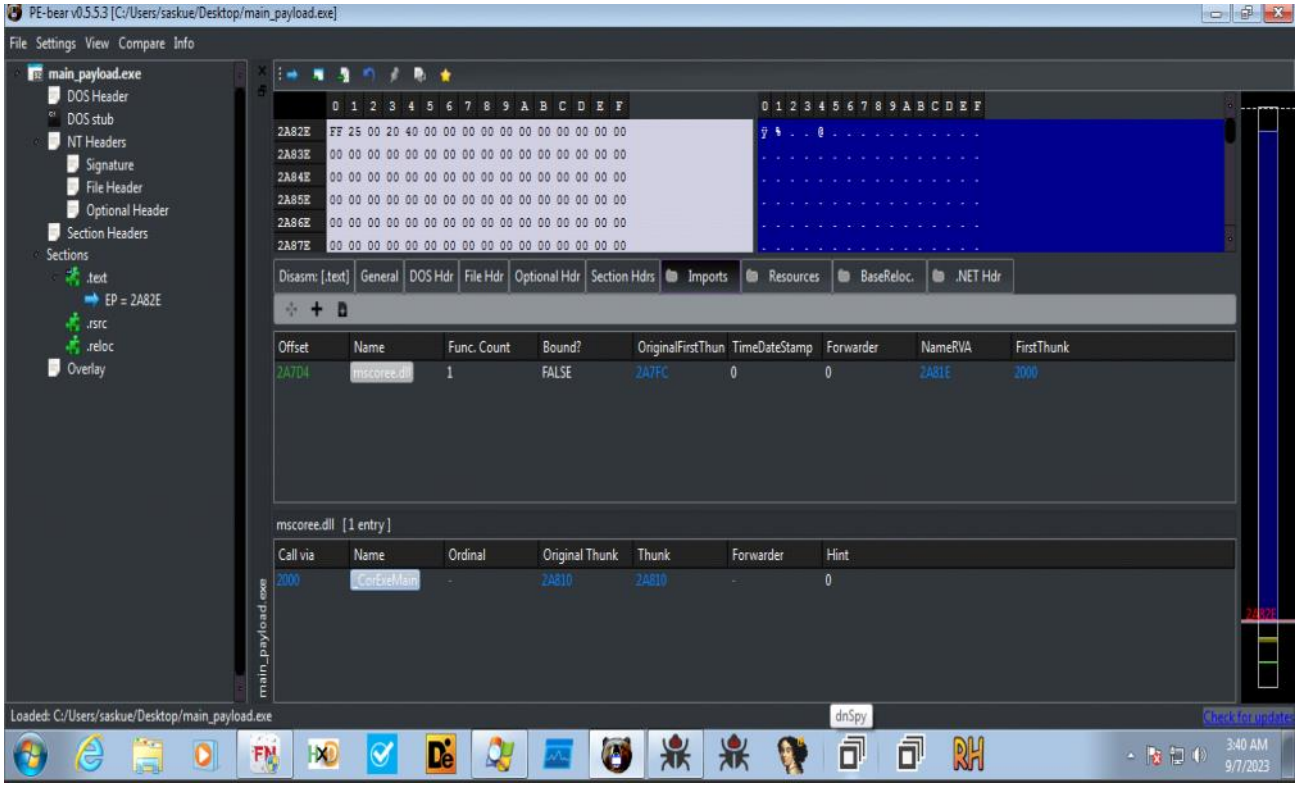
We will get pass the api and will see a PE Written to above mentioned memory region in the remote target process.



Now we will possibly dump the PE from the memory for reversing and analysis of the payload.
After unpacking and dumping the main payload from target process memory I opened it in PEBEAR but import is not getting mapped properly. Time to aligned the file according to process memory.



After aligning the file according to memory we can see imports properly.



Decryption of code:

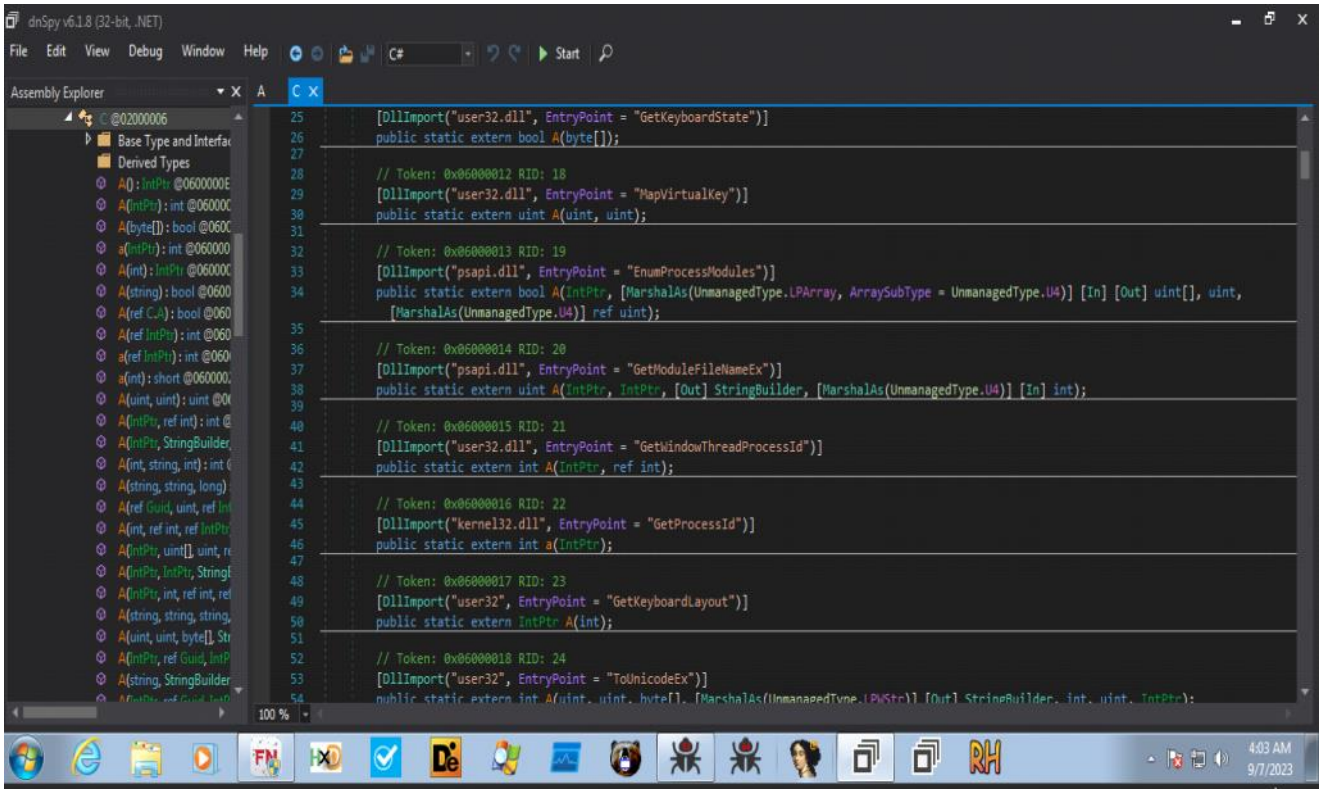
Currently it seems that the main malware is decrypted properly, if we encounter anything encrypted/obfuscated will we decrypt/deobfuscate it either dynamically or via de4dot.

Reversing and analysis:

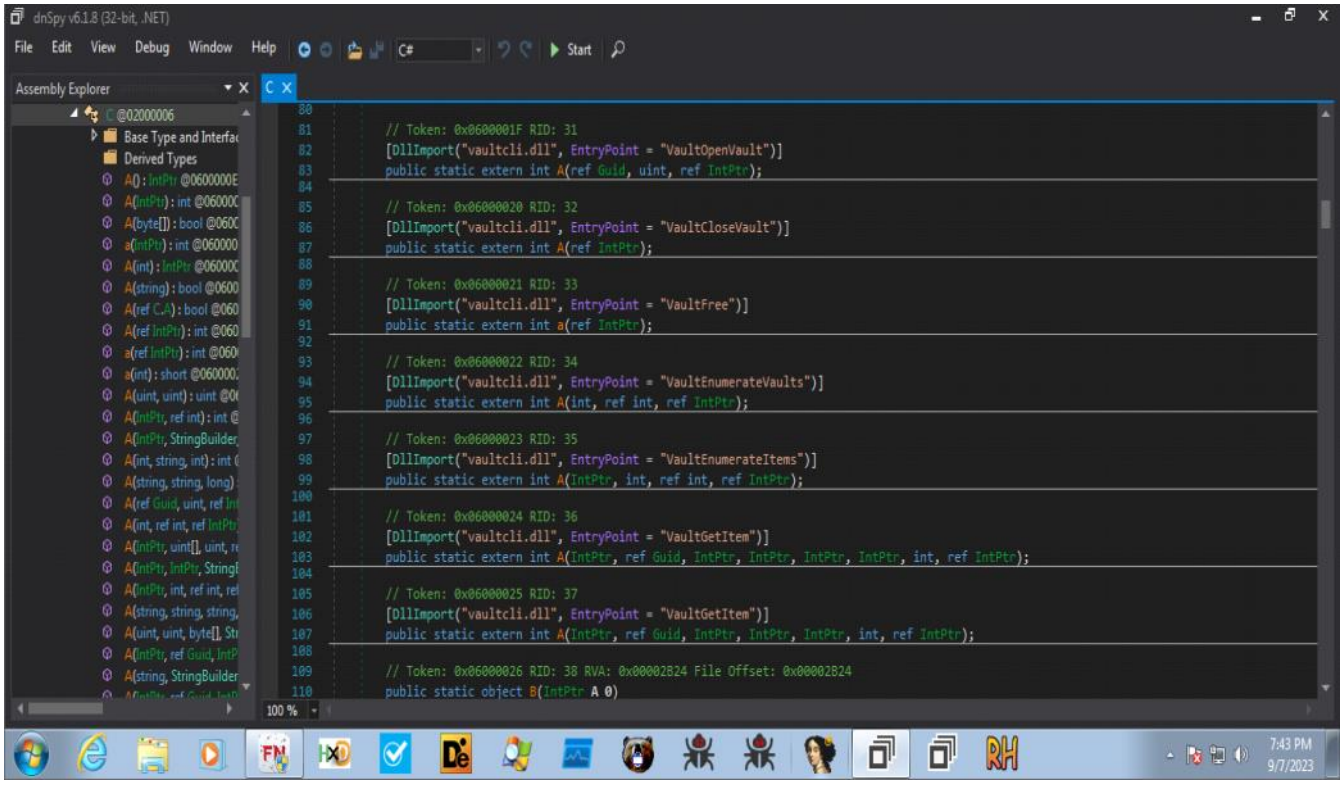
It seems that there is dynamic loading of function, seems interesting as this would bypass from the eye of analyst and detection technology the capabilities the executable has.

Dynamic loading of function:

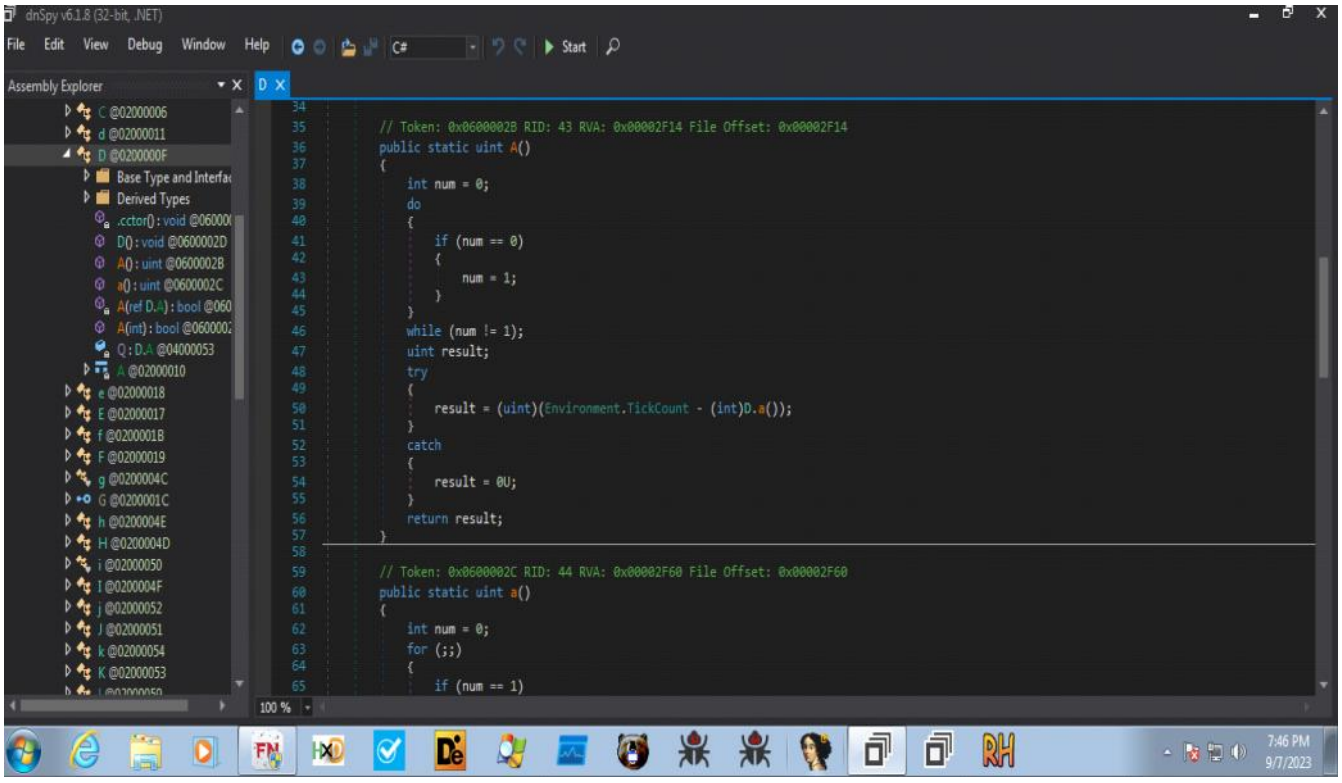
1- GetForegroundWindow from USER32.dll
2- GetWindowText from USER32.dll
3- GetKeyboardState from USER32.dll
4- MapVirtualKey from USER32.dll
5- EnumProcessModules from PSAPI.dll
6- GetModuleFileNameEx from PAPI.dll
7- GetWindowThreadProcessId from USER32.dll
8- GetProcessId from KERNEL32.dll
9- GetKeyboardLayout from user32
10- ToUnicodeEx from USER32
11- GetPrivateProfileString from kernel32.dll



There is also dynamic loading of function from VaultCli.dll such as :
1- VaultOpenVault
2- VaultCloseVault
3- VaultFree
4- Others



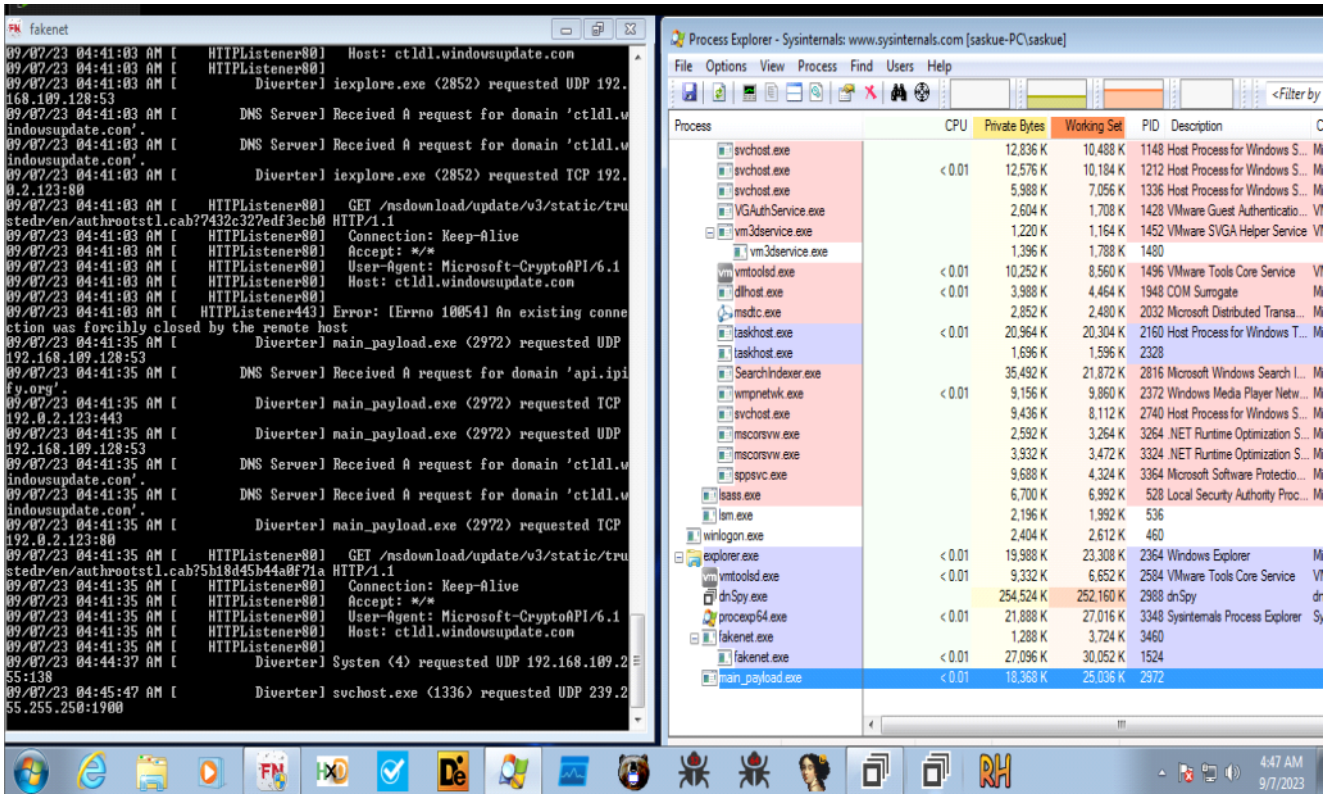
Malware checking for tickcount



To monitor the victim activity windows hooking api are also used:

- SetWindowsHookEx
- CallNextHookEx
- UnhookwindowsHookEx

Malware check for internet connection if its available or not and also tries to get the public ip address of the victim by sending request to api.ipify.org, for internet connection it tries to hide in plain sight by sending web request to CTLDL.WINDOWSUPDATE.COM

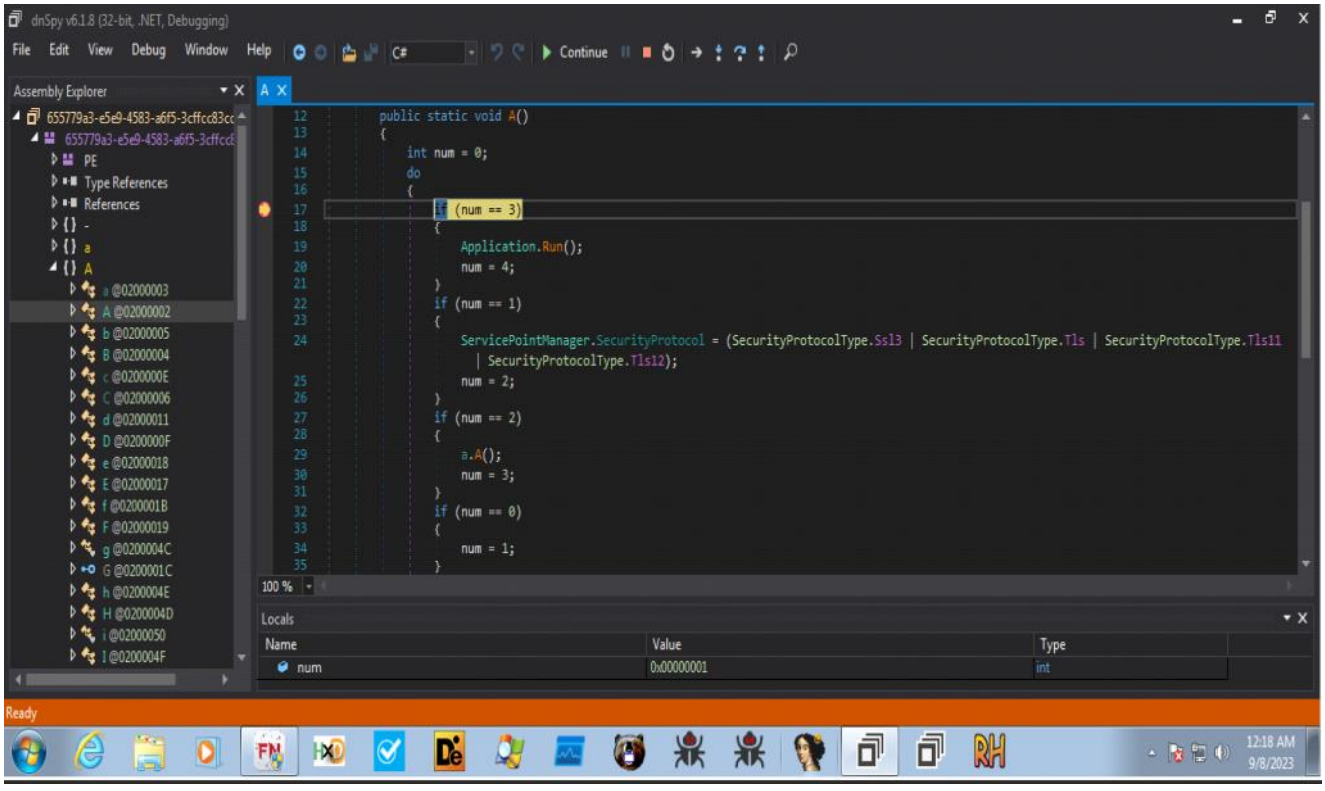


This was just a dry run test of malware. Let's get serious and start dissecting the malware.

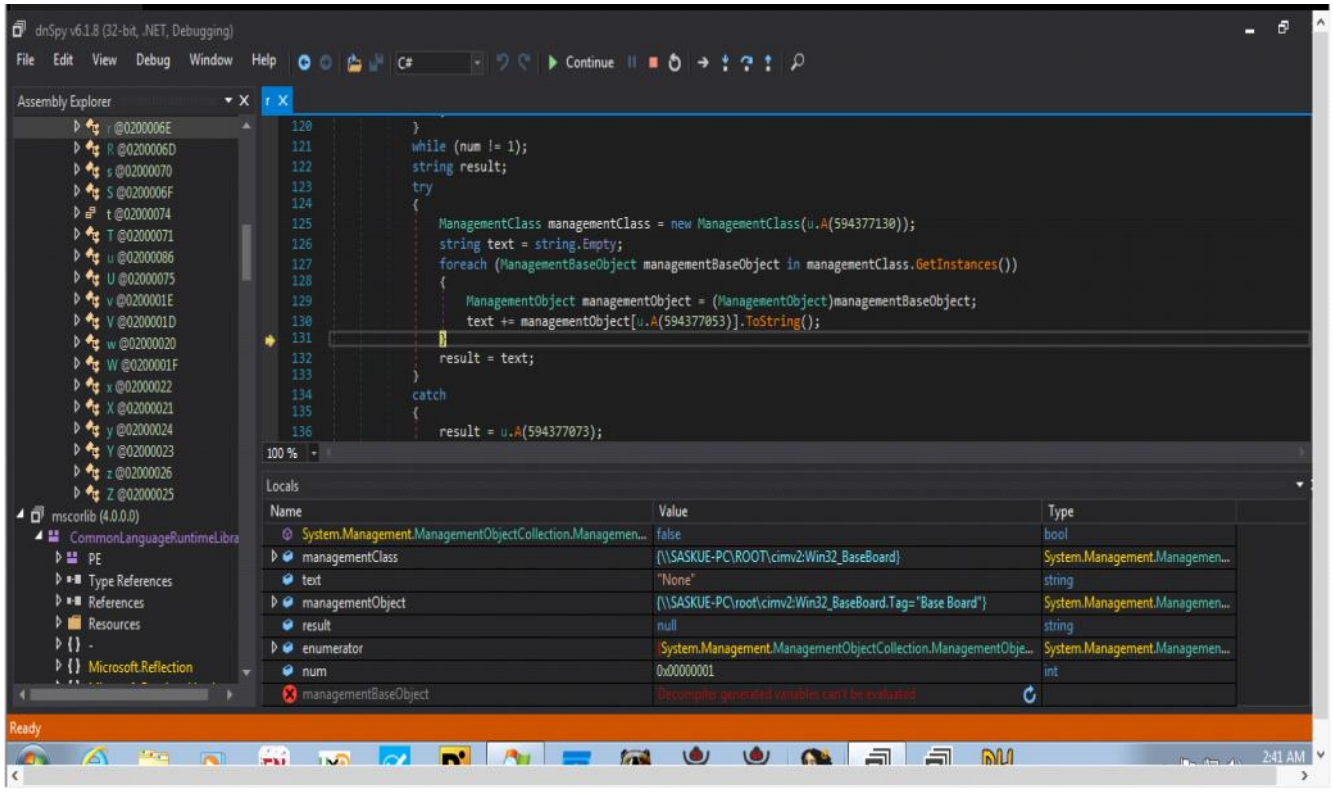
Malware main capabilities:

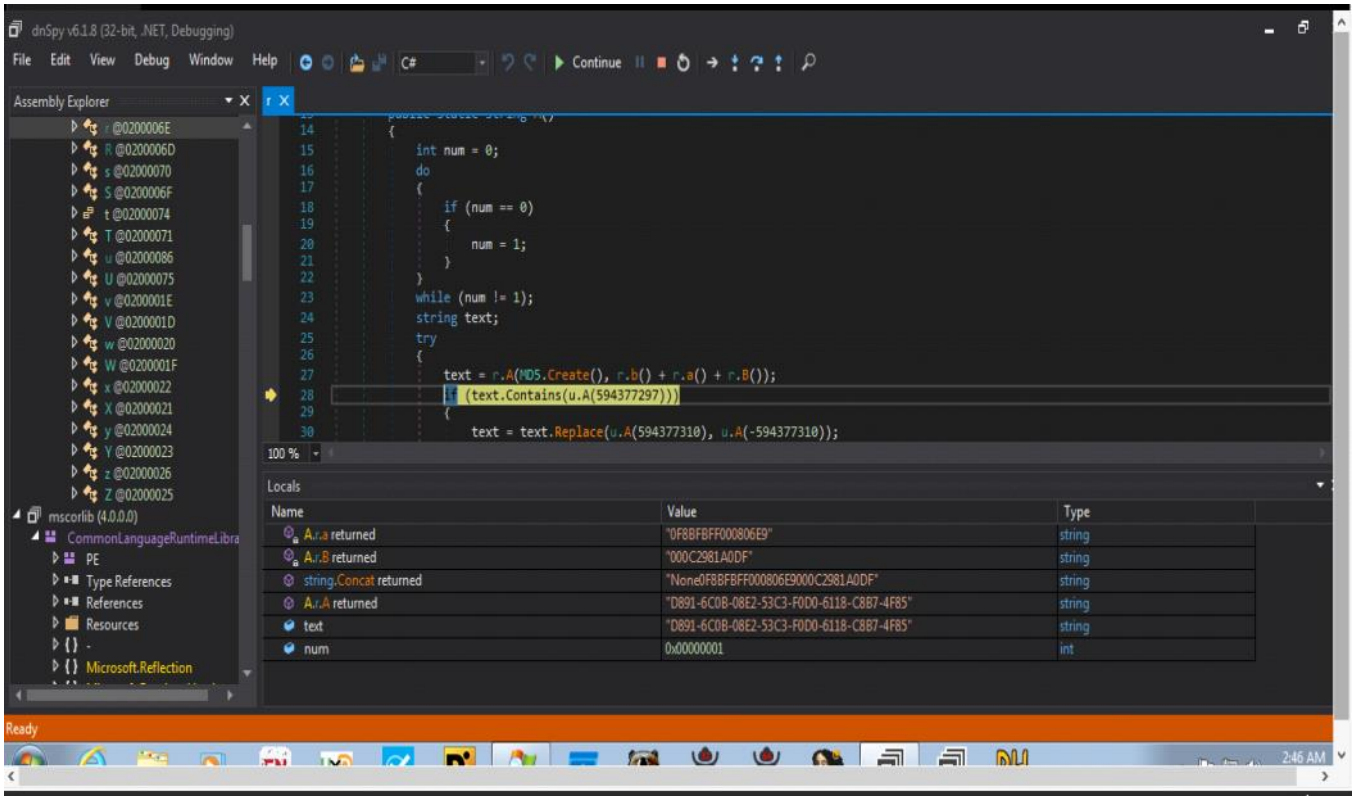
It first loop upto 3 times and during that process it check for Security protocol type :

- SSL3
- TLS
- TLS11
- TLS12

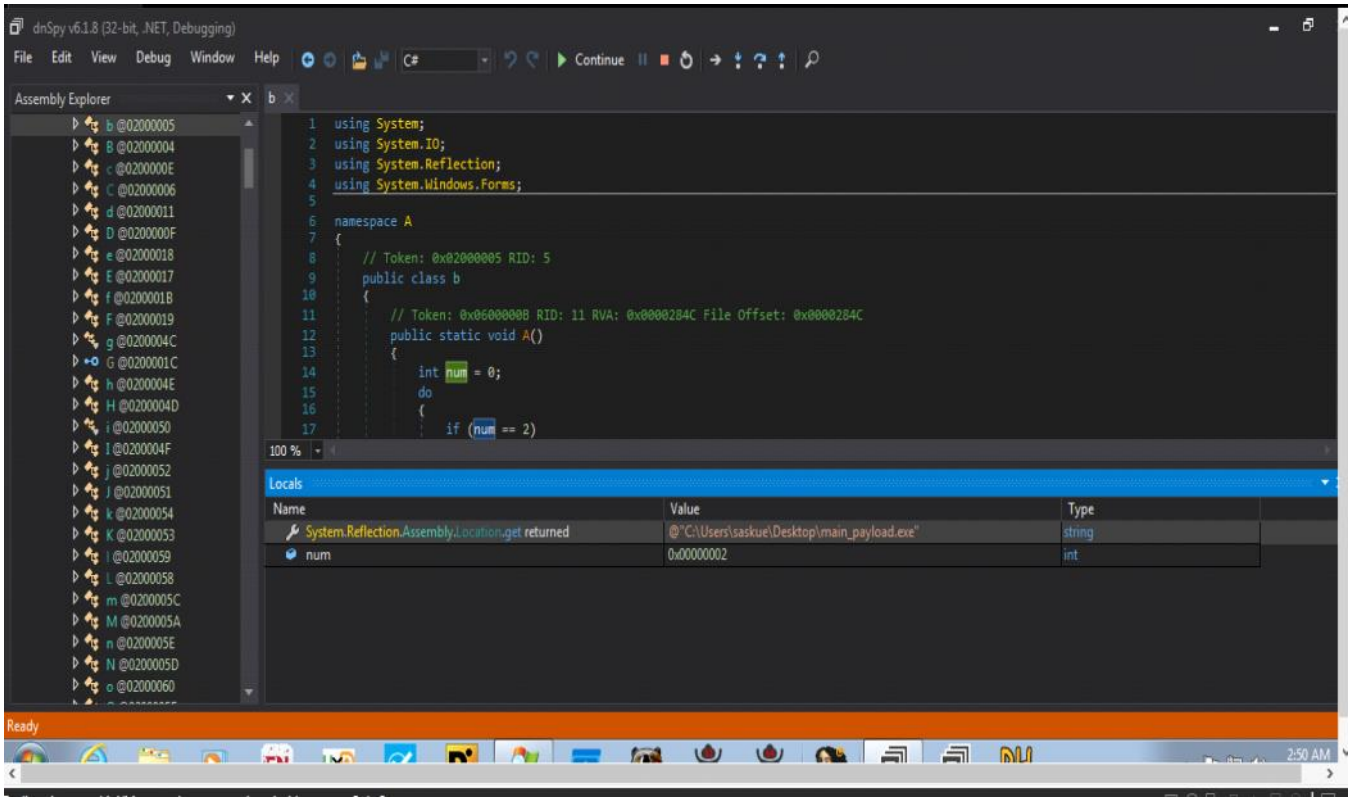


It then create an instance of win32_baseBoard and and start enumerating victim BaseBoard information. Once the function is completed we can see these info being concatenate and then hashed with MD5 to be used for later purpose.

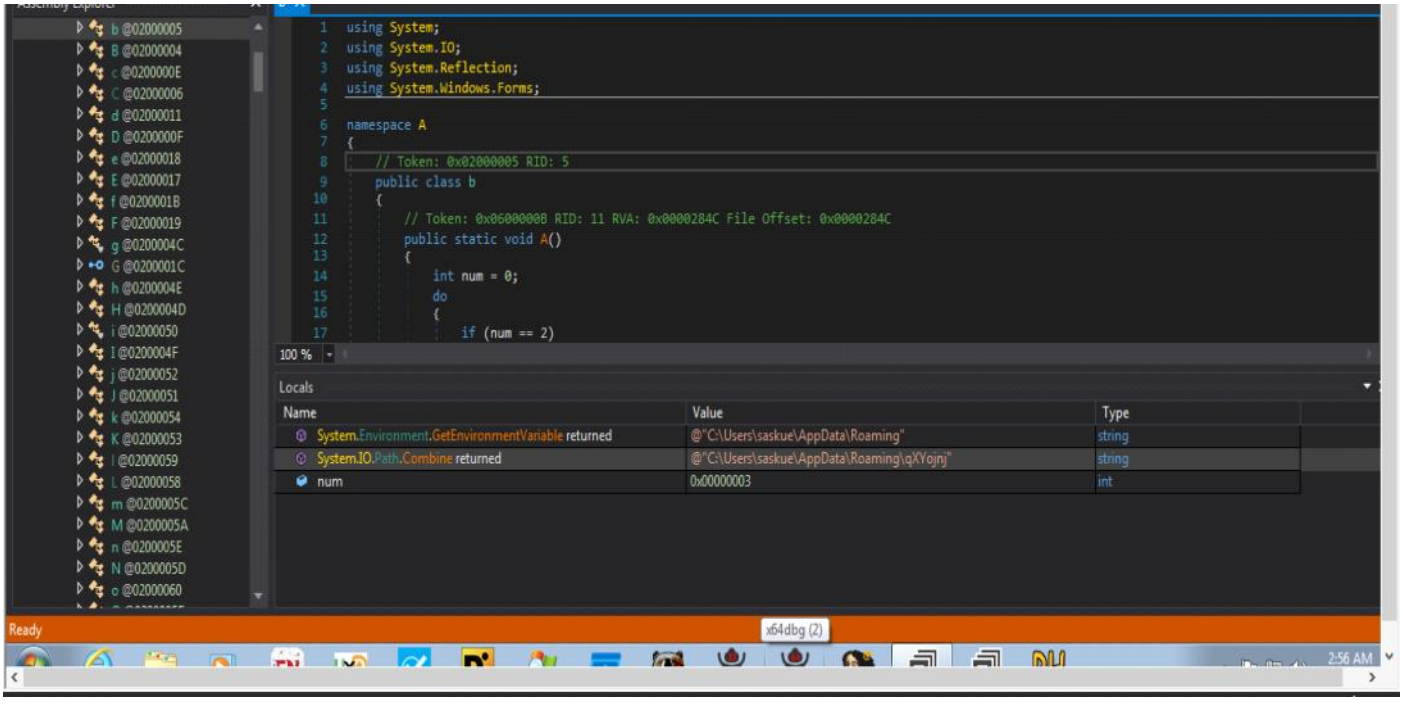




Later on it query for current directory from where malware is running.
Here its : @"C:\Users\saskue\Desktop\main_payload.exe"

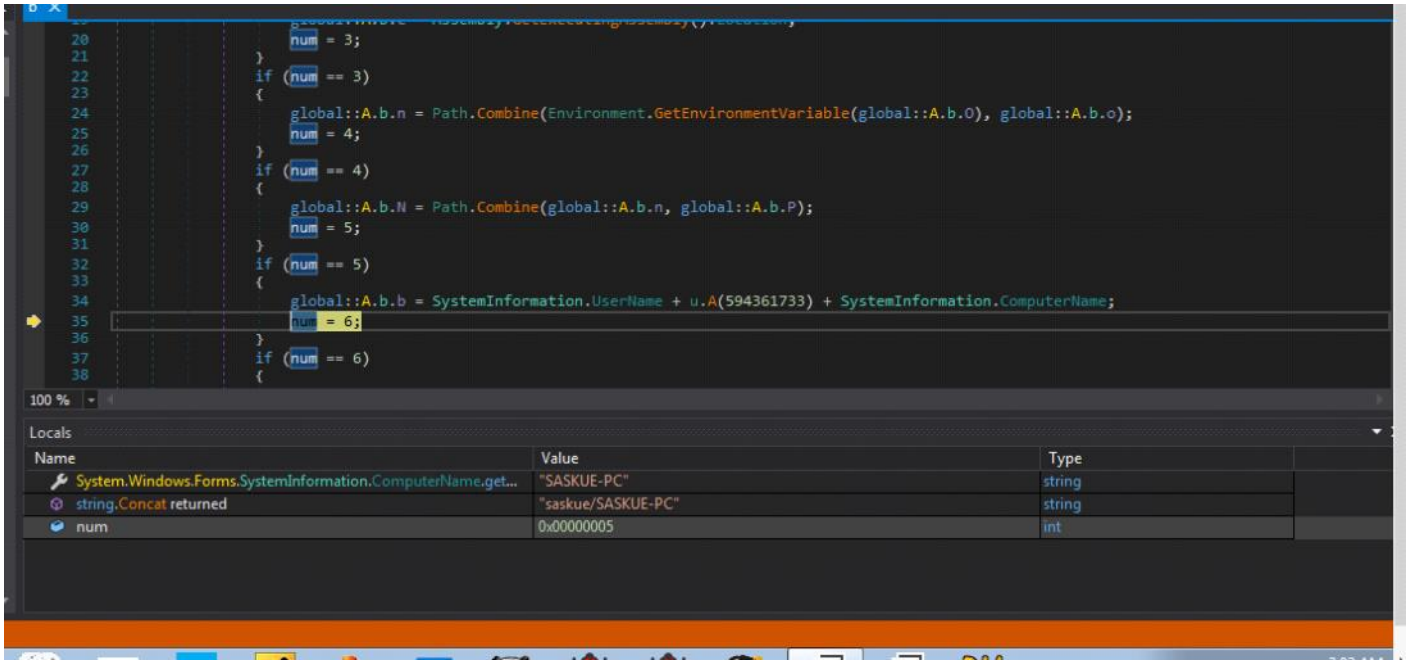


Later on the malware get environment variable and concatenate it with other string which together look like this:
@"C:\Users\saskue\AppData\Roaming\qXYojnj"

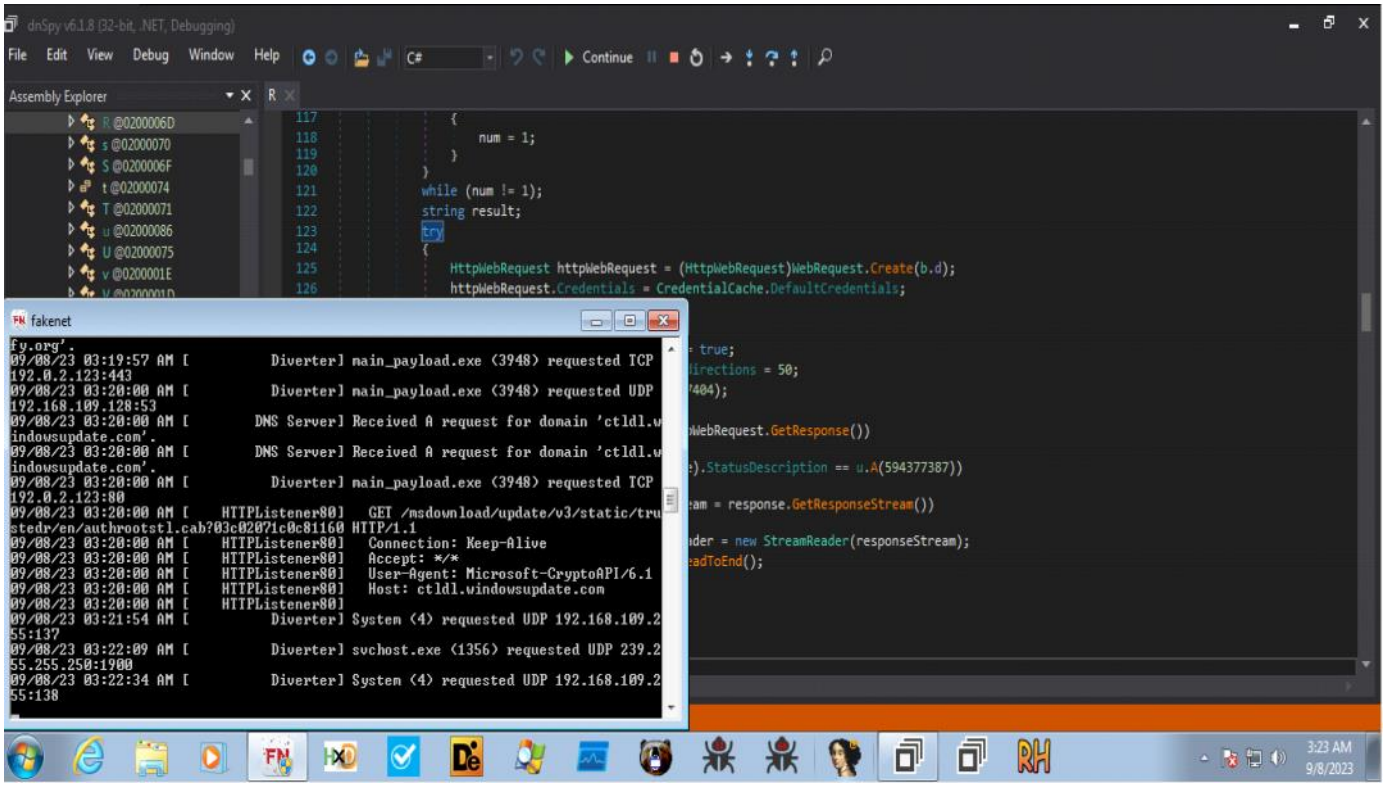


Later on this process I saw a reference for @"C:\Users\saskue\AppData\Roaming\qXYojnj\qXYojnj.exe" seems interesting, let's keep a note of it for while.

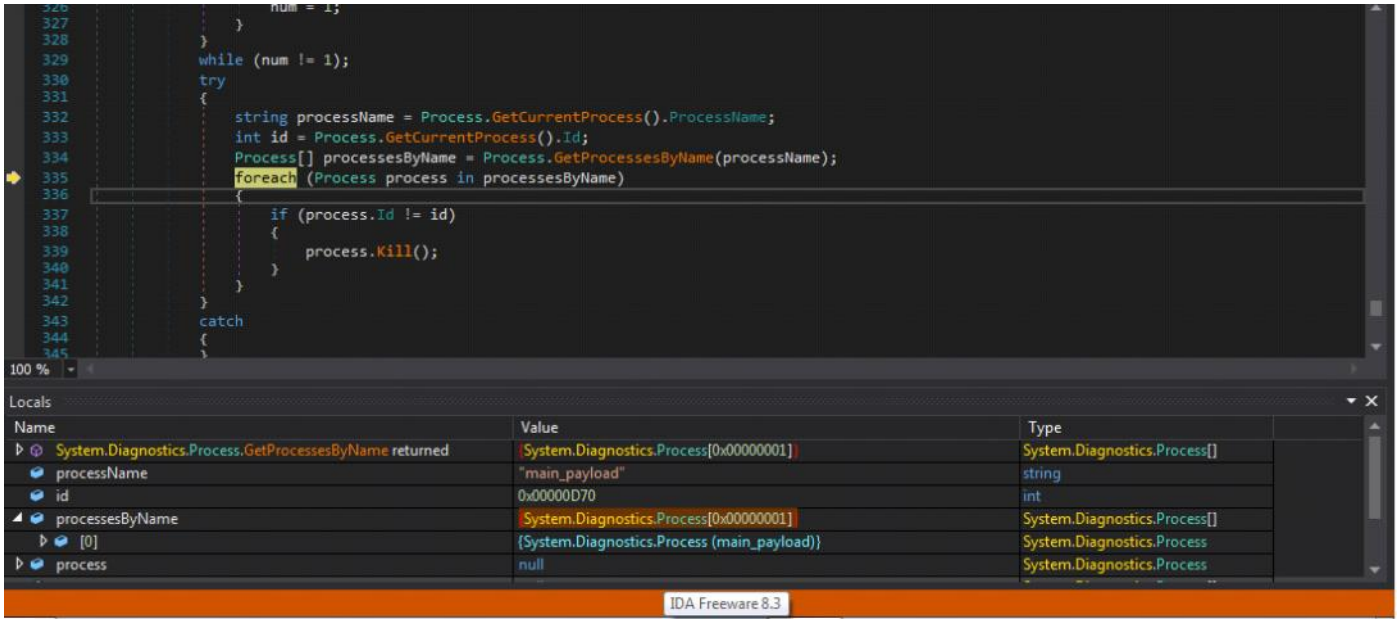
It then enumerate username along with pc-name and concatenate both of the string together.



As I mentioned earlier the malware tries to get public ip info for victim as well as check if the victim have internet access or not.



We could also see the malware getting information about current process and if the current processid doesn't match it will terminate the process.



After recursively going through different function I see an enumeration for opera & Yandex browser which could be interesting too.

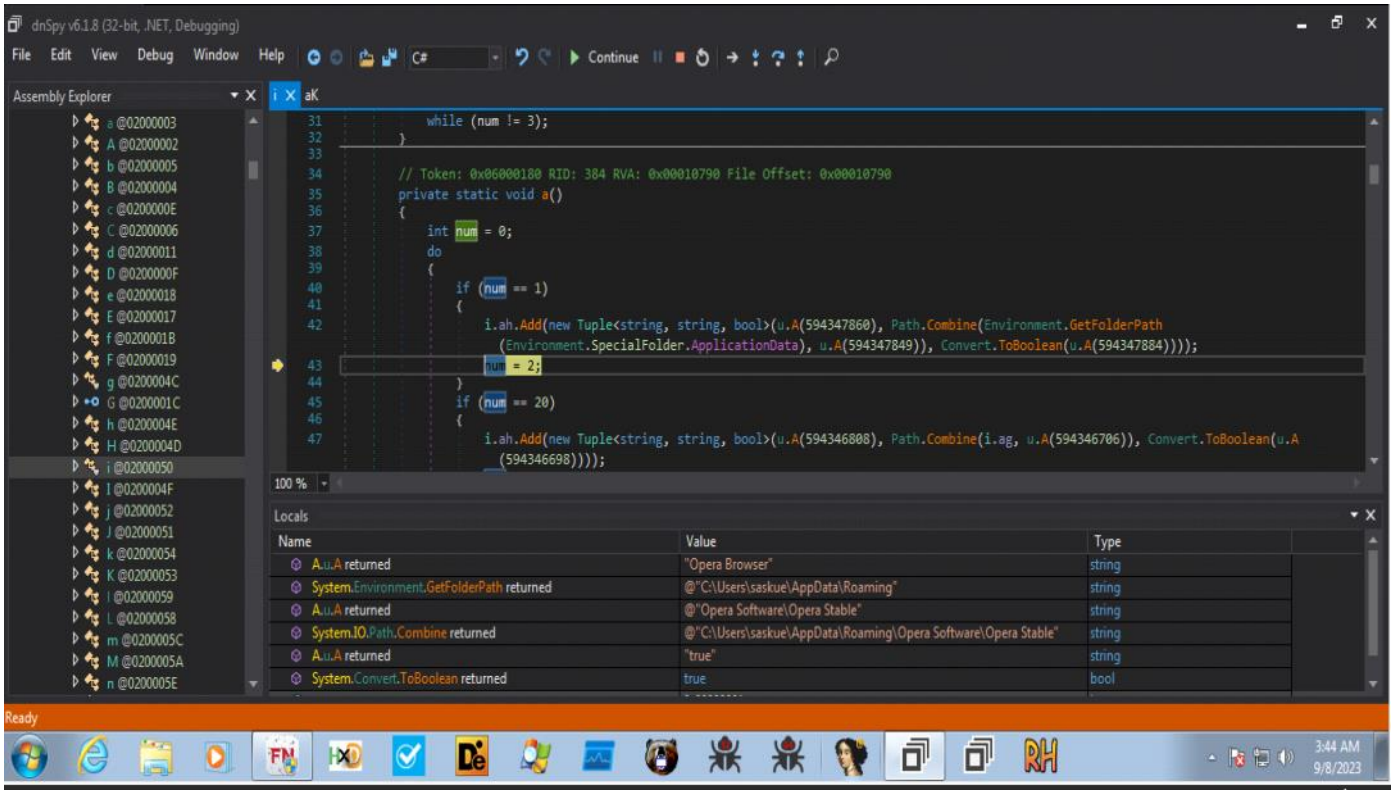
@"C:\Users\saskue\AppData\Roaming\$variable

Here $variable is dynamic.

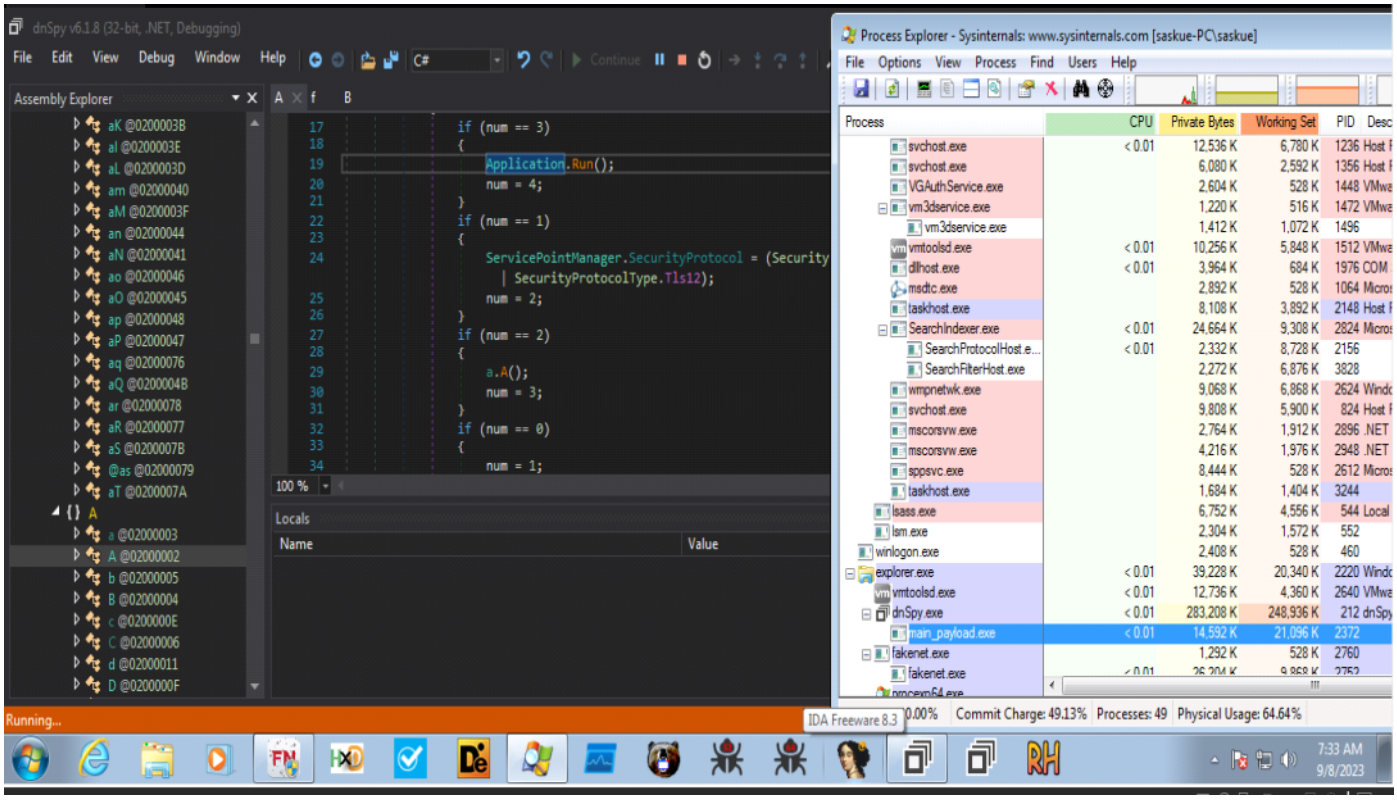
List of browser enumeration (dynamic part of path):

1- OPERA
2- YANDEX
3- IRIDIUM
4- Chromium
5- 7star
6- Torch
7- CoolNovo
8- Kometa
9- Amigo
10- Brave
11- CentBrowser
12- Chedot
13- Orbitum
14- Sputnik
15- Comodo Dragon
16- Vivaldi
17- Citrio
18- 360 browser
19- Uran
20- Liebao browser
21- Elements Browser
22- Epic Privacy
23- Coccoc
24- Sleipnir
25- QIP Surf
26- Coowon
27- Chrome
28- Edge chromium
29- Firefox
30- Seamonkey
31- Thunderbird
32- Blackhawk
33- CyberFox
34- K-meleon

35- IceCat
36- Palemoon
37- IceDragon
38- WaterFox
39- PostBox
40- FlockBrowser
41- UCBrowser



After tracing through different function and coming back to main function, malware runs the application.



There is some dynamic importing of function for clipboard monitor activity's.

- SetClipBoardViewer
- ChageClipboardChain
- Sendmessage

```
using System.Windows.Forms;

namespace A
{
    // Token: 0x02000019 RID: 25
    public class F : NativeWindow
    {
        // Token: 0x0600005E RID: 94
        [DllImport("user32", EntryPoint = "SetClipboardViewer")]
        private static extern IntPtr A(IntPtr);

        // Token: 0x0600005F RID: 95
        [DllImport("user32", EntryPoint = "ChangeClipboardChain")]
        private static extern bool A(IntPtr, IntPtr);

        // Token: 0x06000060 RID: 96
        [DllImport("user32", EntryPoint = "SendMessage")]
        private static extern long A(IntPtr, int, IntPtr, IntPtr);

        // Token: 0x06000061 RID: 97 RVA: 0x00004498 File Offset: 0x00004498
        public void A(F.A A_1)
```

I could also see some cryptographic function too such as bcryptgetproperty etc.

```
using System.Runtime.InteropServices;

namespace A
{
    // Token: 0x02000063 RID: 99
    public static class p
    {
        // Token: 0x060001E3 RID: 483
        [DllImport("bcrypt.dll", EntryPoint = "BCryptOpenAlgorithmProvider")]
        public static extern uint A(out IntPtr, [MarshalAs(UnmanagedType.LPWStr)] string, [MarshalAs(UnmanagedType.LPWStr)] string, uint);

        // Token: 0x060001E4 RID: 484
        [DllImport("bcrypt.dll", EntryPoint = "BCryptCloseAlgorithmProvider")]
        public static extern uint A(IntPtr, uint);

        // Token: 0x060001E5 RID: 485
        [DllImport("bcrypt.dll", EntryPoint = "BCryptGetProperty")]
        public static extern uint A(IntPtr, [MarshalAs(UnmanagedType.LPWStr)] string, byte[], int, ref int, uint);
```

Malware tries to enumerate Microsoft credentials by enumerating files inside the directory @"C:\Users\saskue\AppData\Local\Microsoft\Credentials\" if there is a file found it then add the file name to list otherwise the range is still zero.

```
        // Token: 0x0600008B RID: 139 RVA: 0x000057B0 File Offset: 0x000057B0
        private List<I> A(p.C A_1)
        {
            int num = 0;
            List<I> list2;
            List<string>.Enumerator enumerator;
            for (;;)
            {
                List<string> list;
                if (num == 4)
                {
                    list.AddRange(Directory.GetFiles(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + u.A
                      (594359359)));
```

Locals

Name	Value	Type
System.Environment.GetFolderPath returned	@"C:\Users\saskue\AppData\Local"	string
A.u.A returned	@"\Microsoft\Credentials\"	string
string.Concat returned	@"C:\Users\saskue\AppData\Local\Microsoft\Credentials\"	string
System.IO.Directory.GetFiles returned	{string[0x00000000]}	string[]
this	{A.x}	A.x
value	De	A.p.C
list	Count = 0x00000000	System.Collections.Generic.Lis
Raw View		
Capacity	0x00000000	int

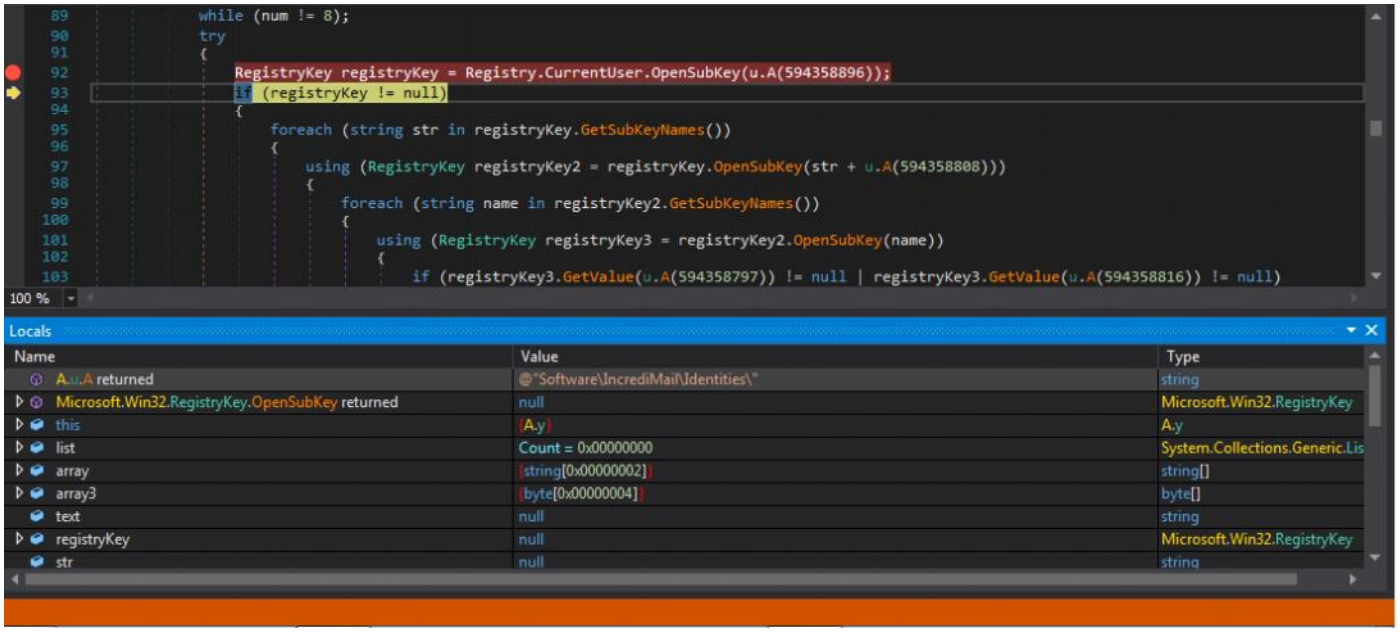
Later on during the process, malware check for falkon profiles directory if it doesn't exists it will get out of directory enumeration function.

```
            for (;;)
            {
                if (num == 4)
                {
                    if (File.Exists(text + u.A(594358569)))
                    {
                        goto IL_88;
                    }
                    num = 5;
                }
                if (num == 2)
                {
                    text = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + u.A(594358576);
                    num = 3;
```

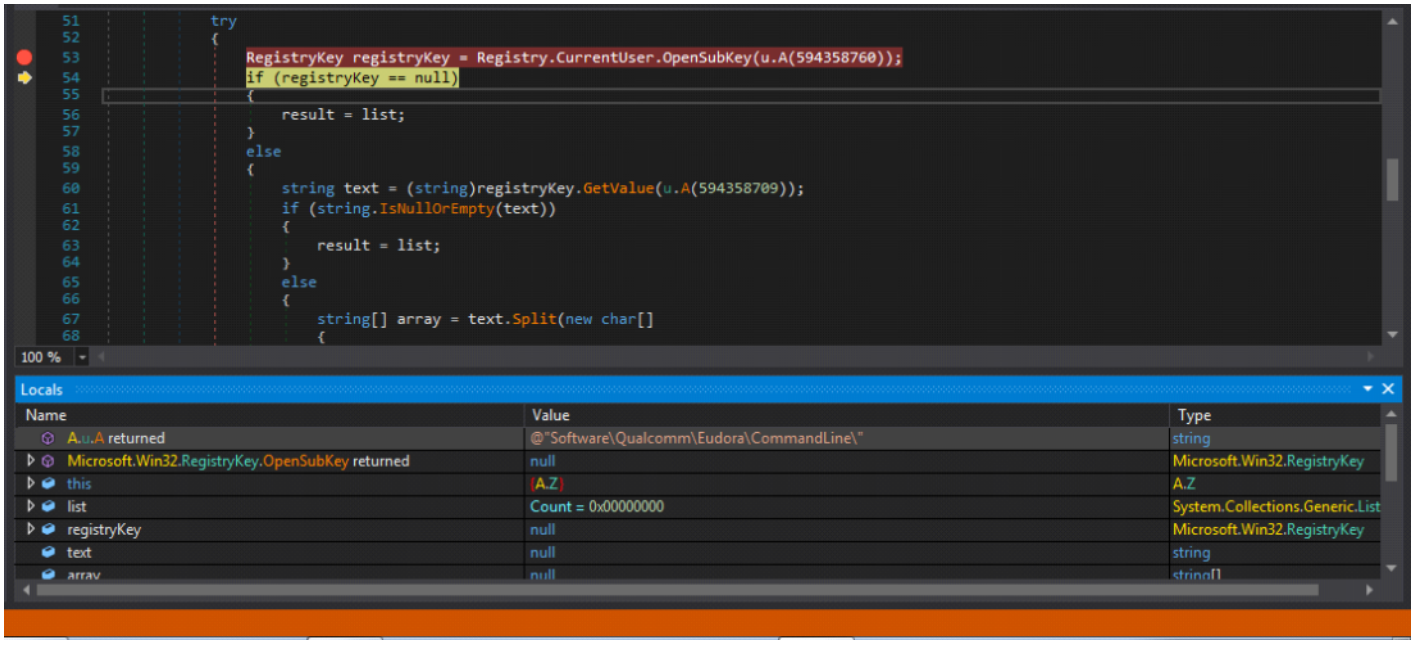
Locals

Name	Value	Type
System.Environment.GetFolderPath returned	@"C:\Users\saskue\AppData\Local"	string
A.u.A returned	@"\falkon\profiles\"	string
string.Concat returned	@"C:\Users\saskue\AppData\Local\falkon\profiles\"	string
this	{A.z}	A.z
list	Count = 0x00000000	System.Collections.Generic.Lis
text	@"C:\Users\saskue\AppData\Local\falkon\profiles\"	string
value	null	string
text2	null	string
k	null	A.k

Malware check for availability of sending mail by enumerating registry keys (subkeynames and get the value for the subkey), @"Software\IncrediMail\Identities\".

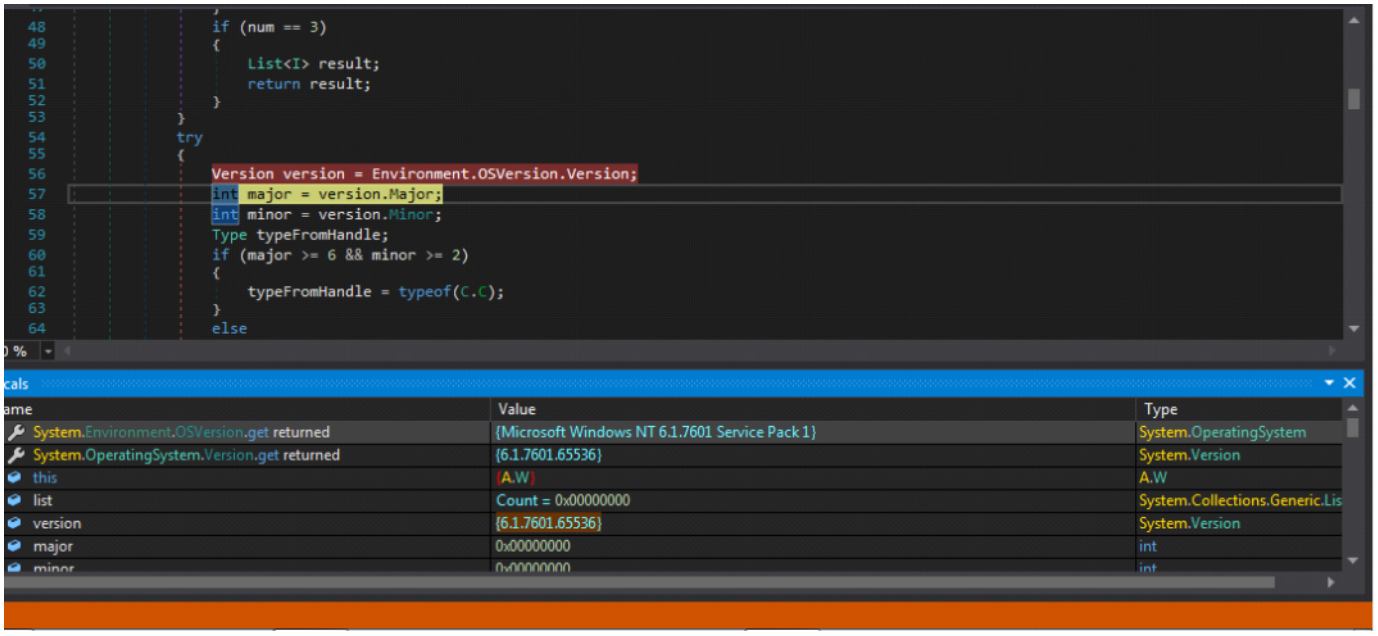


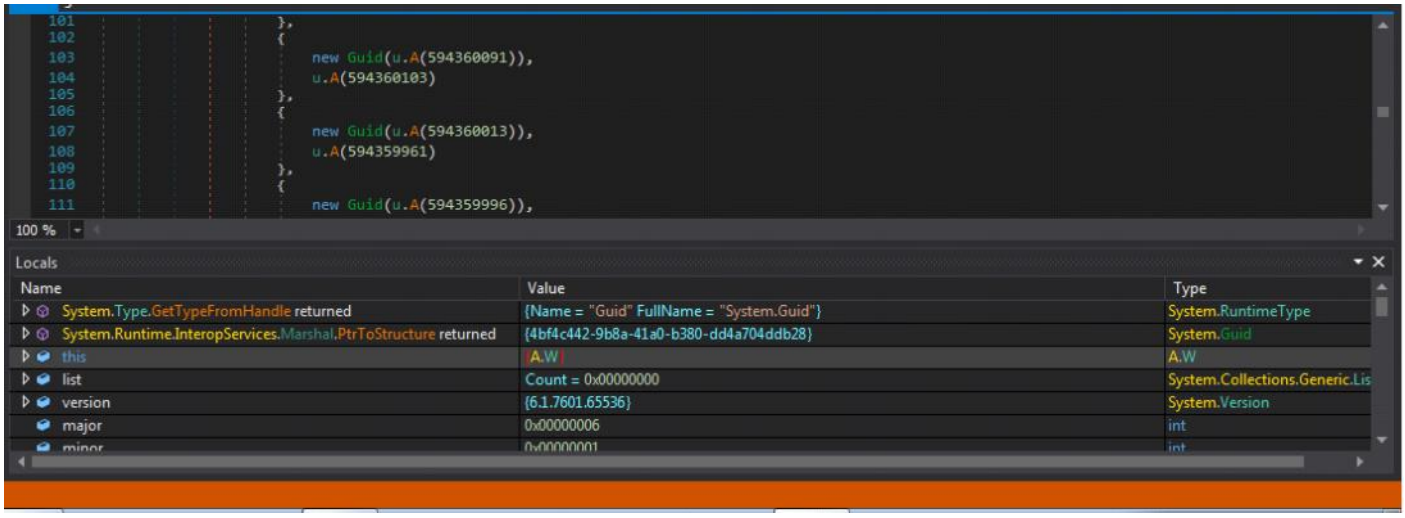
It also check for availability of eudora email client from qualcomm, maybe this information will be used later on the stage by attacker. @"Software\Qualcomm\Eudora\CommandLine\" as well as claws mail.



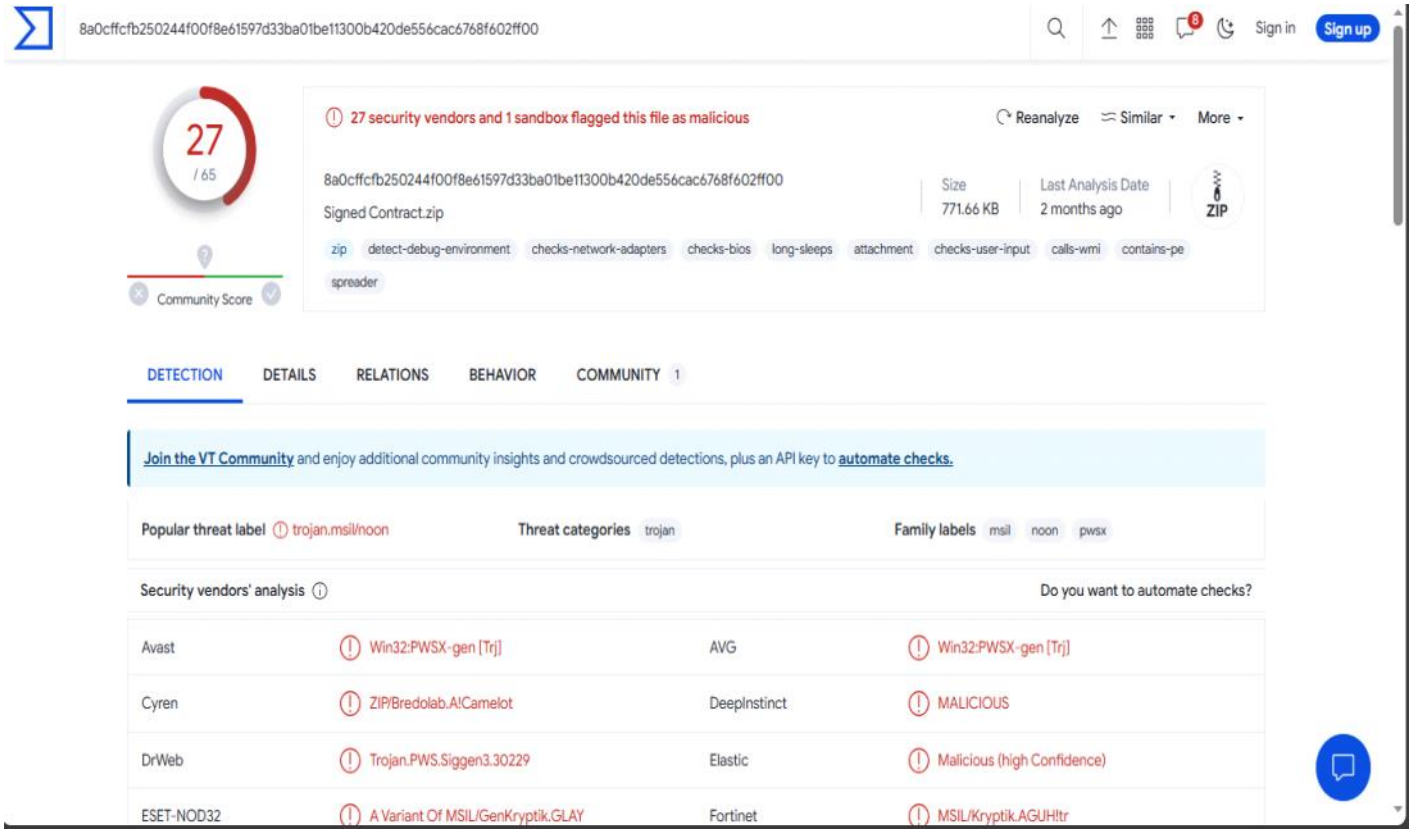
There is also enumeration of system information such as:

1- OS Version
2- System GUID





Virustotal also detect it as malware with 27 vendor detecting it as malicious.



Tries to make outbound connection to these domain name:

api.ipify.org
api.telegram.org
fp2e7a.wpc.2be4.phicdn.net
fp2e7a.wpc.phicdn.net
api4.ipify.org

Detection content:

Summary of Functionality:

1- Enumeration system information
2- Dynamic Loading of function
3- Checking of tickcount
4- Capture of keystrokes
5- Vault operation and enumeration
6- Hooking for monitoring operation
7- Internet connectivity checks
8- Query systemBoard (win32_baseboard)
9- Browser stealing activities
10- Email stealing activities (including credentials)
11- Capturing clipboard operation
12- Stealing microsoft windows credentials
13- C2 Communication

Indicators of compromise:

MD5 Hash ==> A8D2EBA5D6F81FFA035C5B7633C3D543
MD5 Hash ==> 7B9B91CAABD2288177ADB28DFC05259E
MD5 Hash ==> C58DAB11EBC43EFF532D1C5960BBEA19
MD5 Hash ==> 621CC12C5F9582078B54E2F06D73897C
MD5 Hash ==> 747DEF87C68D8D562ED2DF3D670AC14B
IP ==> 173.231.16.76:443

IP ==> 149.154.167.220:443
IP ==> 64.185.227.155:443
MD5 Hash ==> BBBF758D4C186330042CAC5BAE2F380A
MD5 Hash ==> 24021D5B93A4CAC6C2BA3308F8F28600
MD5 Hash ==> 57C9A7F6337D9A54DAB712E5BA55502B

YARA RULES:

Malware-Signature/Agent Tesla/agentesla.yar at main · Deepanjalkumar/Malware-Signature (github.com)

agentesla.yar

```
rule agent_tesla
{
    meta:
        author ="Deepanjal Kumar"
        description="Agent tesla is a malware which is a stealer capable of stealing information such as microsoft crd
                    keystrokes as well apart from stealing browser creds and email sensitive info"
        org ="Operation Falcon"
        date="9/9/2023"
        md5="A8D2EBA5D6F81FFA035C5B7633C3D543"
        sha256="7F52A1B5ED77D701595F1DE3543C7C4567B521EFCAF5EA12EEA93F55BA1203D1"

    strings:
        $binary="655779a3-e5e9-4583-a6f5-3cffcc83cda2.exe"
        $privatedetails="<PrivateImplementationDetails>{3F199F7E-B77D-4E68-8CDF-CDCEEA1BE4D1}"
        $strangestring="$2785b0b4-42ac-406a-b6aa-e12e31686fb4"
        $strangestringnew="69754B7763534E"
        $emailaddr="tranquochung252001@gmail.com"

    condition:
        any of them

}
```

SIGMA RULES:

Malware-Signature/Agent Tesla/agentesla.yml at main · Deepanjalkumar/Malware-Signature (github.com)

agentesla.yml

```yaml
title: agent tesla
ruletype: Sigma
author: Deepanjal kumar
date: 2023/09/09
description: Agent tesla is a malware which is a stealer capable of stealing information such as microsoft crdentials
    keystrokes as well apart from stealing browser creds and email sensitive info
detection:
    SELECTION_1:
        EventID: 4688
    SELECTION_2:
        Channel: Security
    SELECTION_3:
        Hashes:
        - '*MD5=A8D2EBA5D6F81FFA035C5B7633C3D543*'
        - '*SHA256=7F52A1B5ED77D701595F1DE3543C7C4567B521EFCAF5EA12EEA93F55BA1203D1*'
        - '*IMPHASH=F190ED098182C8C4D9A5B5E866BC9CED*'
    SELECTION_4:
        md5: 7B9B91CAABD2288177ADB28DFC05259E
    SELECTION_5:
        sha256: 6E86F0BBACD63A35C78B533CC2C223374D5E2CA9938169A6C7A312211494F728
    SELECTION_6:
        Imphash: 2916DDA3C80B39A540B60C072A91A915
    condition: ((SELECTION_1 and SELECTION_2) and ((SELECTION_3) or
        (SELECTION_4 or SELECTION_5 or SELECTION_6)))
falsepositives:
- Unknown
```

Mapping to MITRE ATTACK:

TECHNIQUE NAME	TECHNIQUE ID
Windows management instrumentation	ID: T1047
Sandbox evasion	ID: T1497
Credential from password stores	ID: T1555
Account discovery	ID: T1087
Clipboard data	ID: T1115
Browser information discovery	ID: T1217