

# Reversing and Technical Analysis of Agent Tesla:

## Introduction:

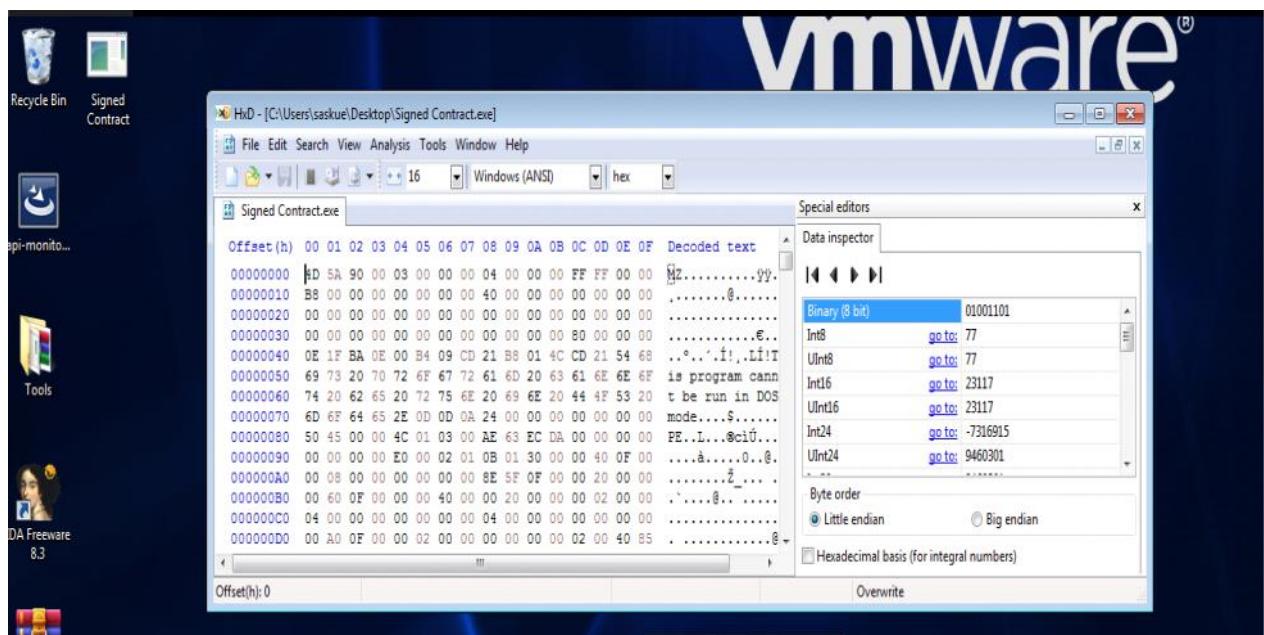
Agent tesla is a malware which is RAT with functionality dealing with stealing sensitive information from victim computer. It is written in .NET and collect information such as keystrokes and login credentials, just like my previous paper on reversing agniane stealer. It was first seen in wild year 2014.

## 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.

pestudio 9.31 - Malware Initial Assessment - www.winitor.com

file settings about

functions (773)

namespace (21)	blacklist (2)	ordinal (0)	library (1)
btnThemNCC_Click	-	-	mscoree.dll
btnSuaNCC_Click	-	-	mscoree.dll
btnXoaNCC_Click	-	-	mscoree.dll
btnTimKiemNCC_Click	-	-	mscoree.dll
btnHienThiNCC_Click	-	-	mscoree.dll
btnLamMoiNCC_Click	-	-	mscoree.dll
dgvNCC_CellContentClick	-	-	mscoree.dll
Supplier_Load	-	-	mscoree.dll
get_ResourceManager	-	-	mscoree.dll
get_Culture	-	-	mscoree.dll
set_Culture	-	-	mscoree.dll
get_iuKwvSN	-	-	mscoree.dll
get_Default	-	-	mscoree.dll
get_User	-	-	mscoree.dll
set_User	-	-	mscoree.dll
get_Password	-	-	mscoree.dll
set_Password	-	-	mscoree.dll
get_RememberMe	-	-	mscoree.dll
set_RememberMe	-	-	mscoree.dll
get_LayChamCong	-	-	mscoree.dll
set_LayChamCong	-	-	mscoree.dll
get_LayTenNhanVien	-	-	mscoree.dll
set_LayTenNhanVien	-	-	mscoree.dll
get_ThemChamCong	-	-	mscoree.dll

sha256: 7F52A1B5ED77D701595F1DE3543C7C4567B521EFCAF5EA12EEA93F55BA1203D1

cpu: 32-bit file-type: executable subsystem: GUI entry-point: 0x000F5F8E signature: Microsoft .NET

9:42 PM 9/6/2023

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.

pestudio 9.31 - Malware Initial Assessment - www.winitor.com

file settings about

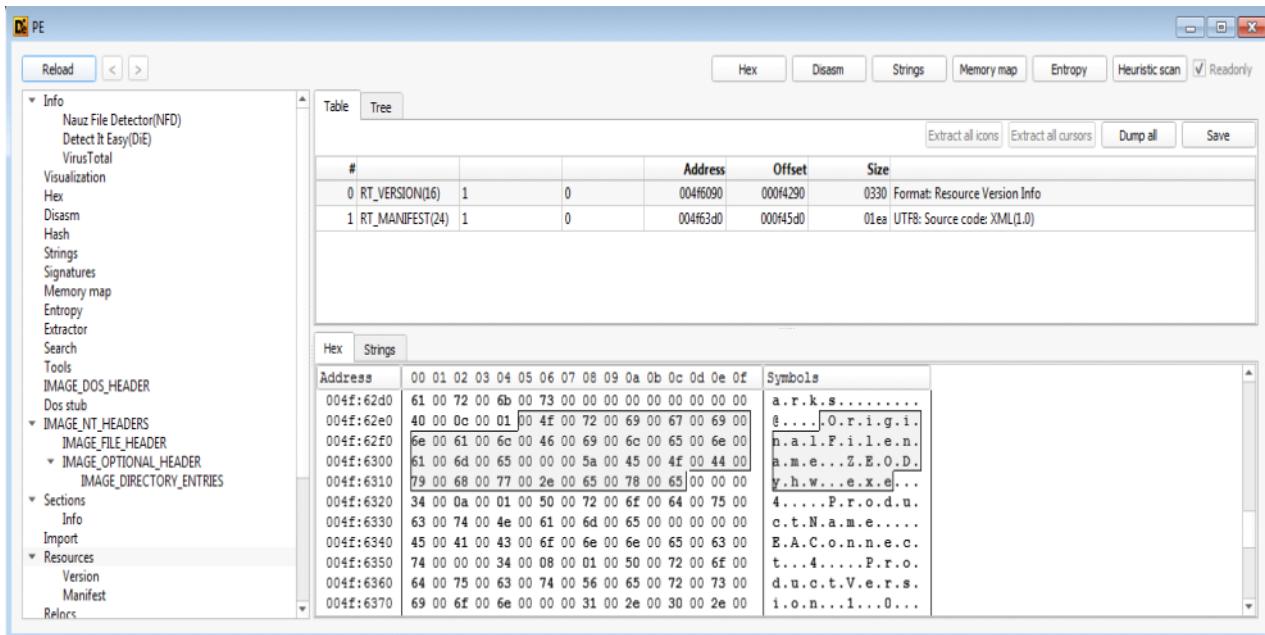
property value

md5-1	3864BE75A5FDA2BF246D0C018D113D86
sha1	7A0C9FD938A3706577D9A5C4CF2E3B4690CC6368
sha256	CBC6655B6D020887588E2A2C68E4ACD39981B4EA9AE21F742EDB34272FA4C58
age	1
size	36 bytes
format	RSDS
debugger-stamp	0xC2B19448 (Tue Jul 04 18:15:34 2073   UTC)
path	ZEODyhw.pdb
Guid	4ED4A4BD-89D8-48CD-83F6-61F07FE096E7

sha256: 7F52A1B5ED77D701595F1DE3543C7C4567B521EFCAF5EA12EEA93F55BA1203D1

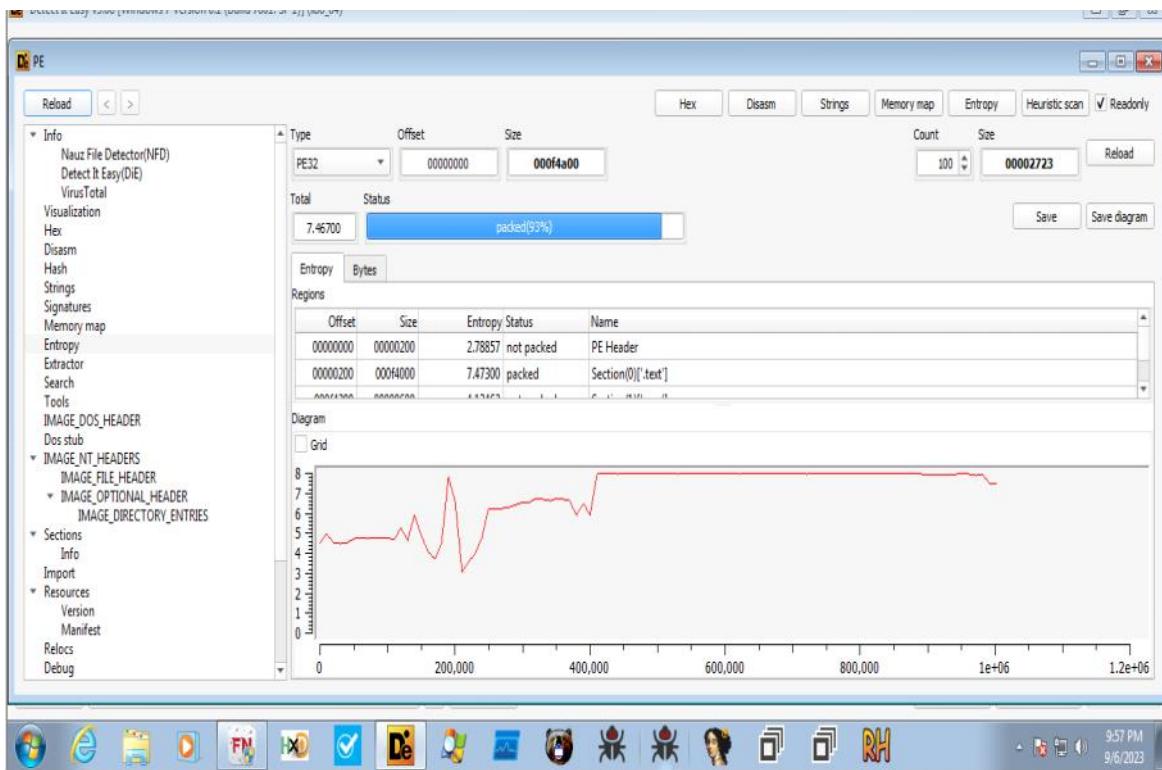
cpu: 32-bit file-type: executable subsystem: GUI entry-point: 0x000F5F8E signature: Microsoft .NET

9:49 PM 9/6/2023



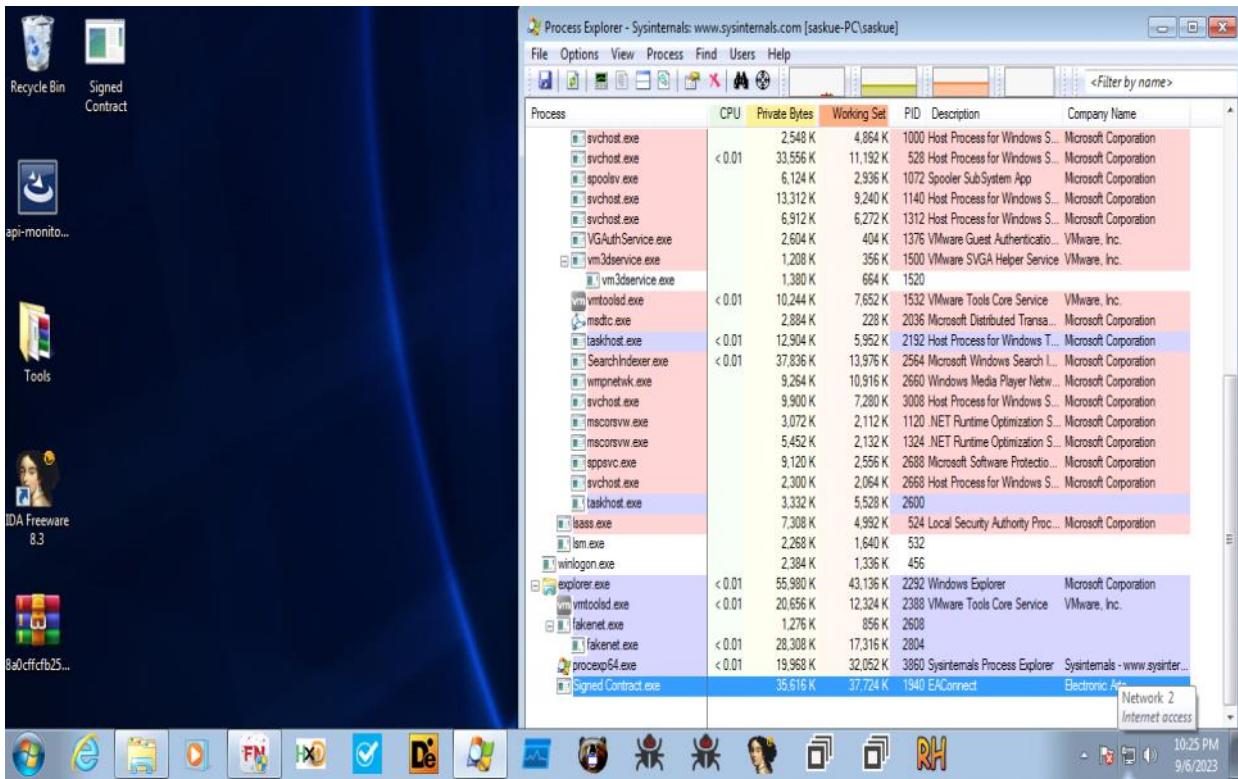
DIE detect it also as .NET executable and also considered it as packed too so there is not much to do except unpack the next payload for further analysis

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.

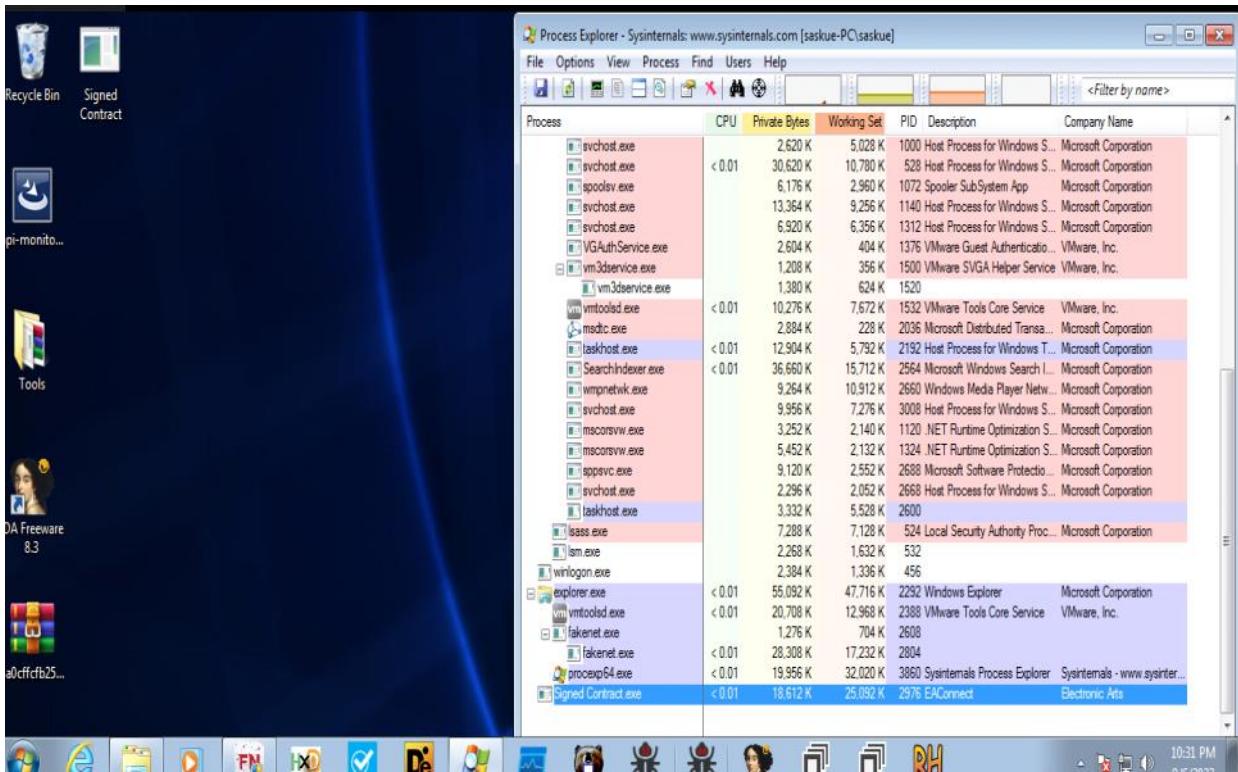


### Initial Dynamic Analysis:

For the initial Dynamic Analysis I will stick to Process Explorer & ProcMon to observe 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.



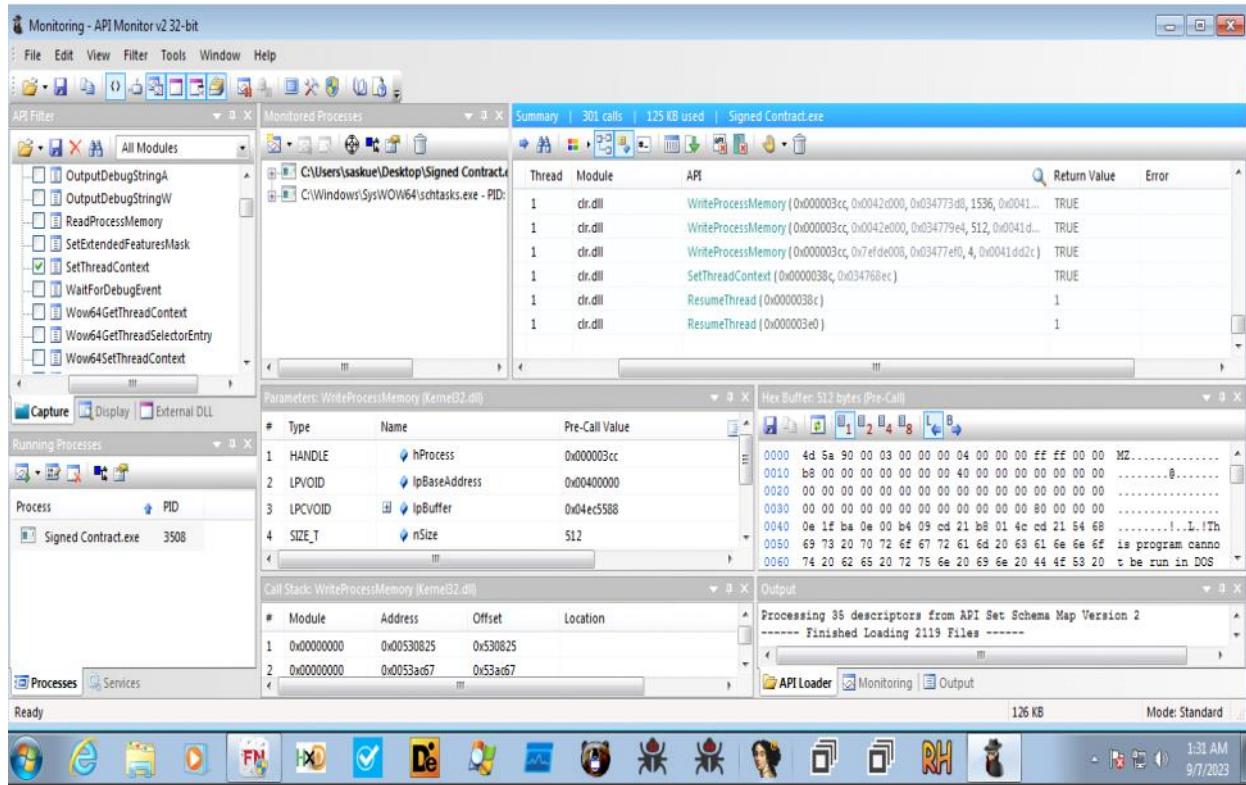
Now that we know the flow of unpacking here is how I am gonna unpack the main malware and extract it, for this we are going to use x64dbg and will monitor the api :

- 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 is packed and written to remote remote process by these api in order:

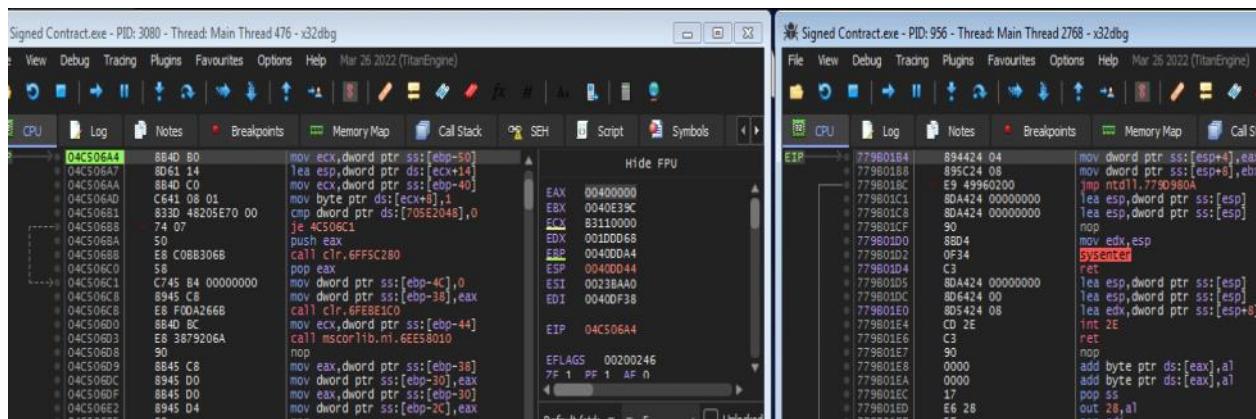
- 1- VirtualAlloc
- 2- CreateProcessA
- 3- VirtualAllocEx
- 4- WriteProcessMemory
- 5- SetThreadContext
- 6- ResumeThread

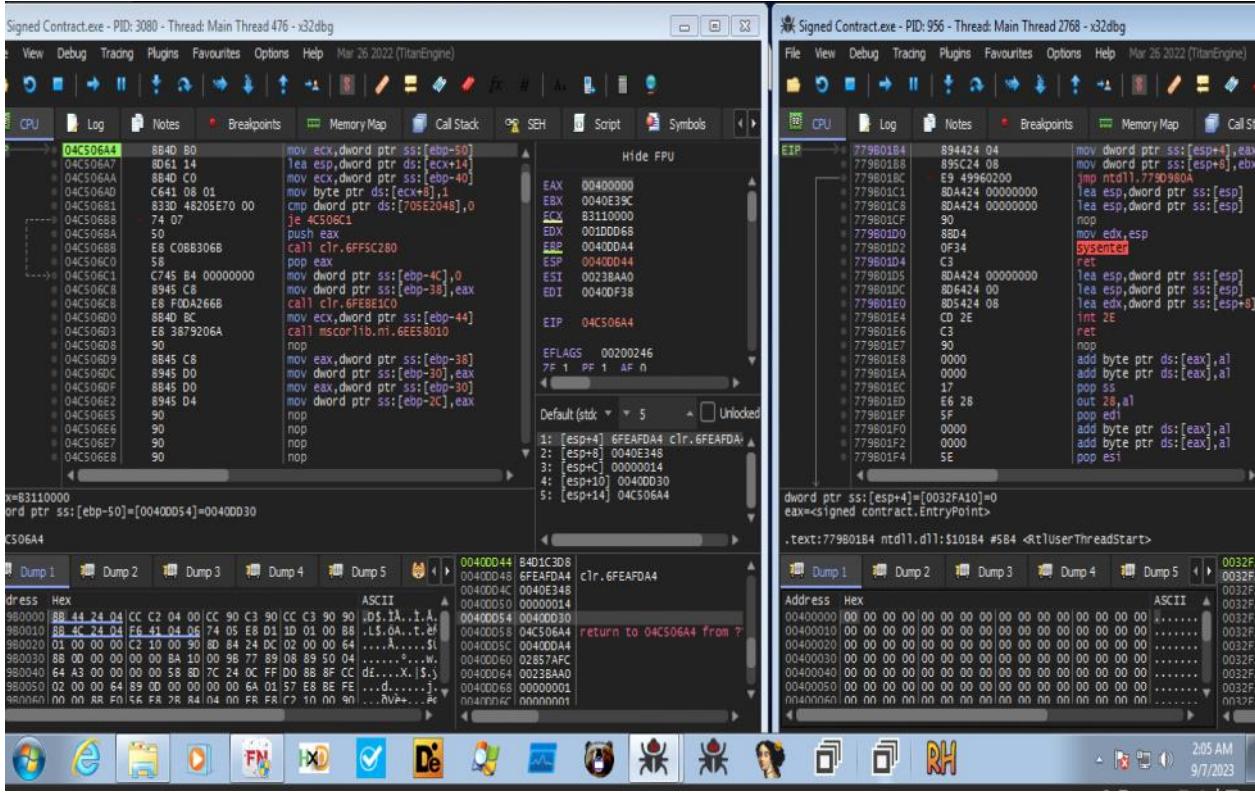


Unpacking of the malware:

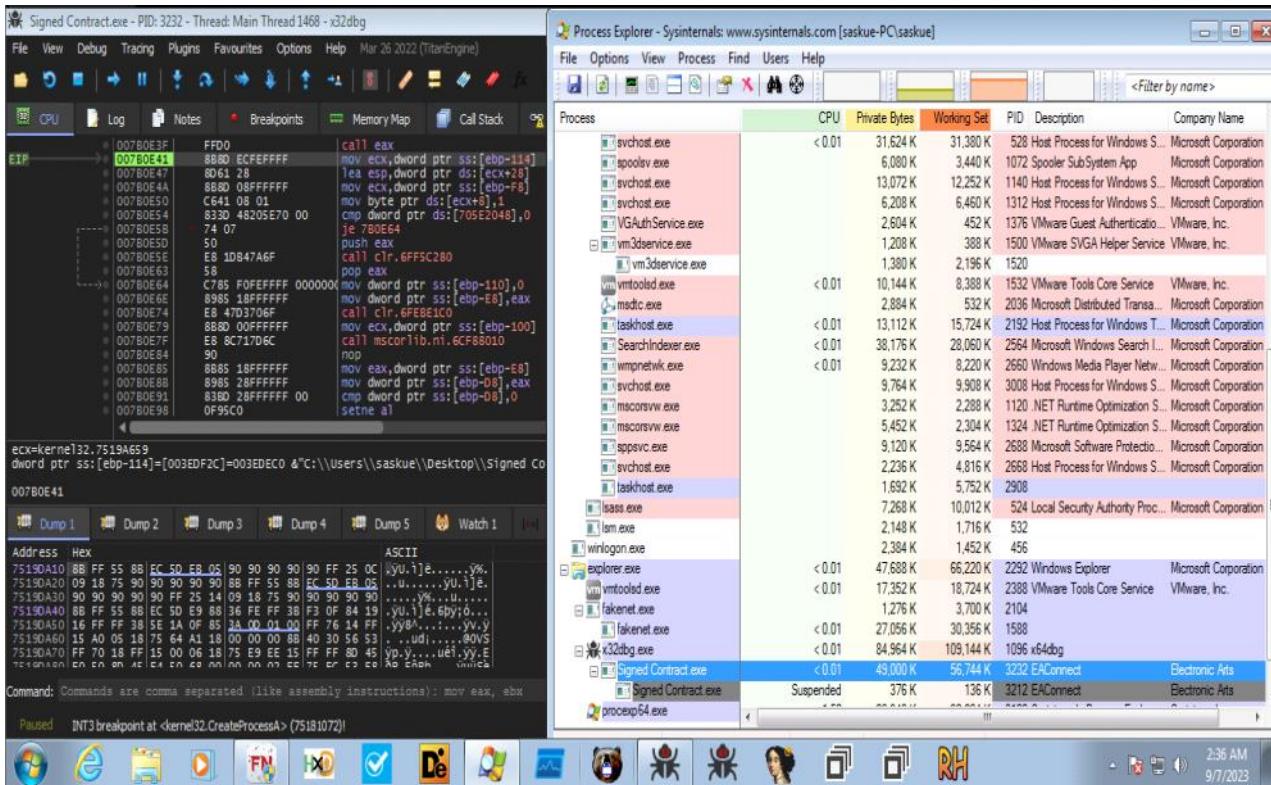
For the unpacking of the malware we are going to use x32dbg and will set breakpoint on appropriate api and will extract the payload once the whole malware is mapped into memory.

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.

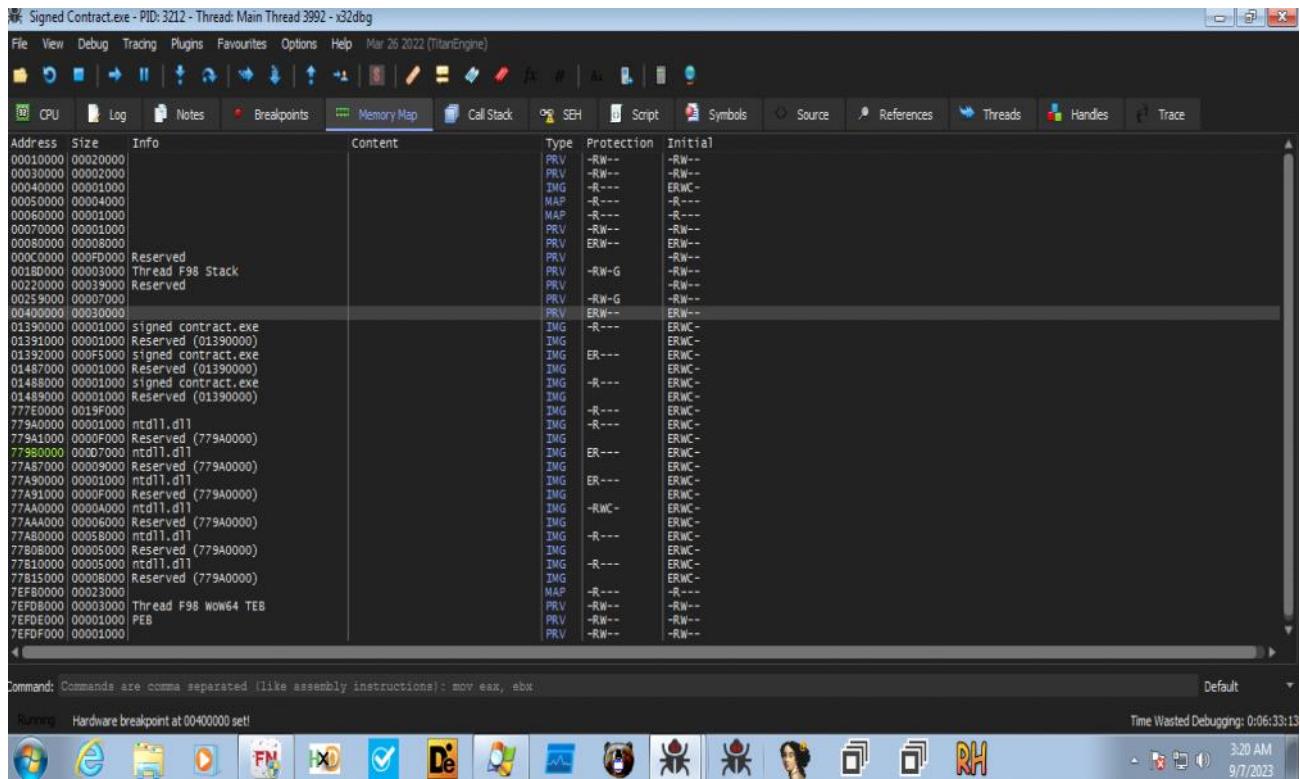




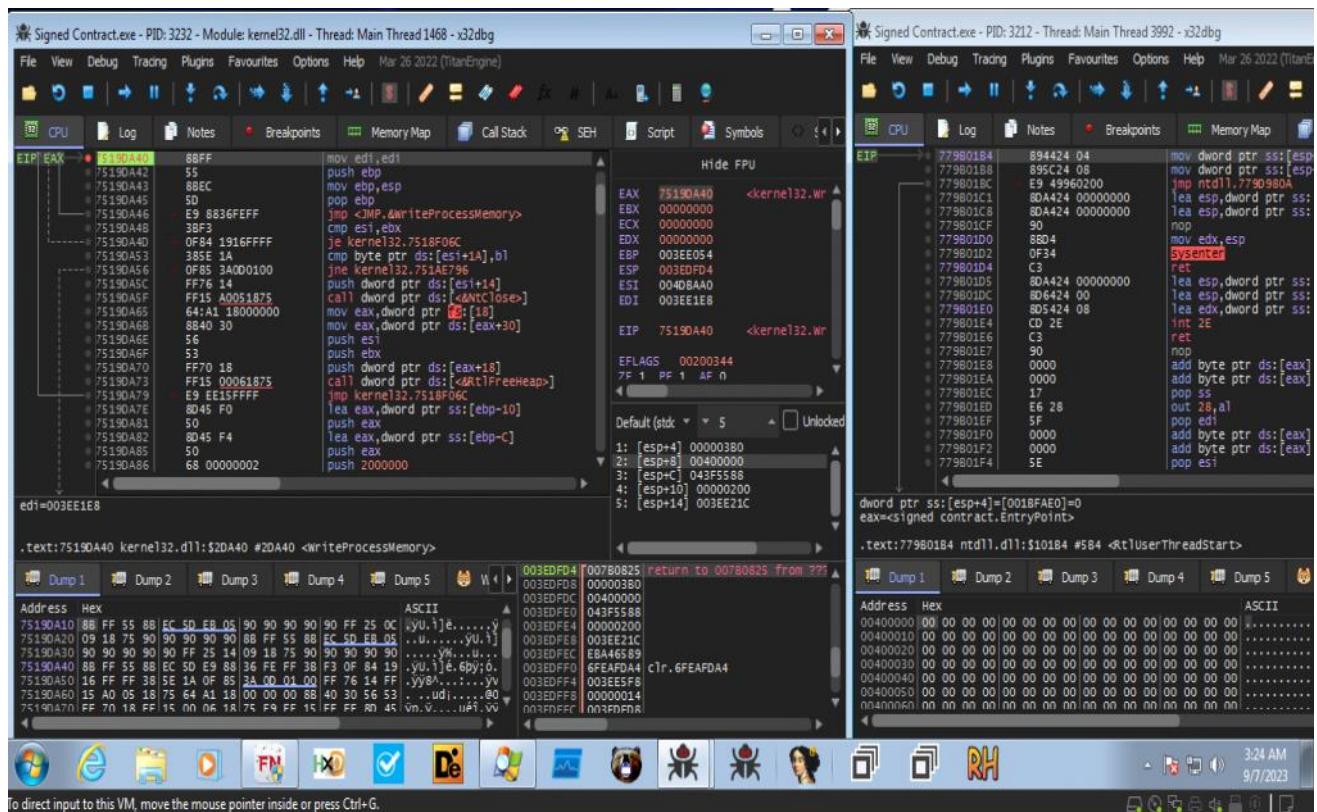
You can see that a new process is created in suspended state too.



If we look at memory map of the newly created process we will observe 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.



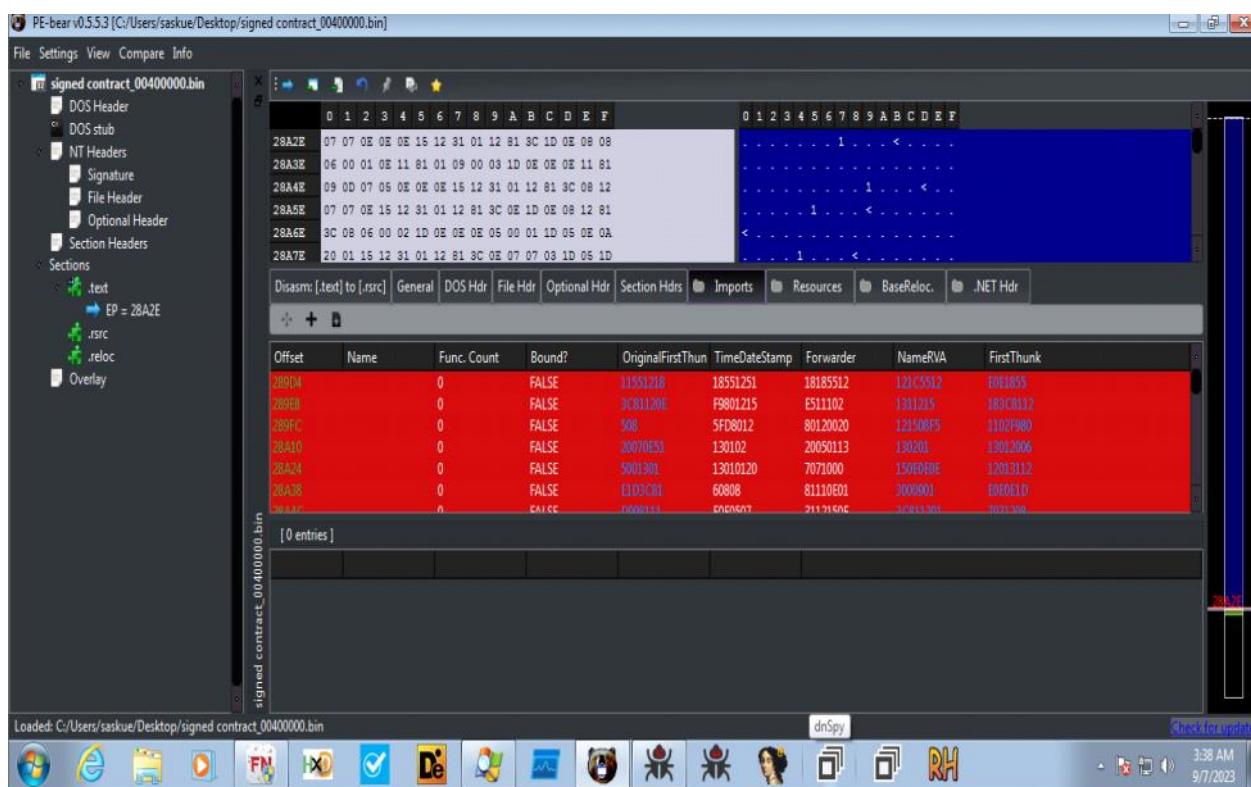
As soon as we hit the writeprocessmemory api we could see it being used to write in that same particular memory region for the newly created remote process.



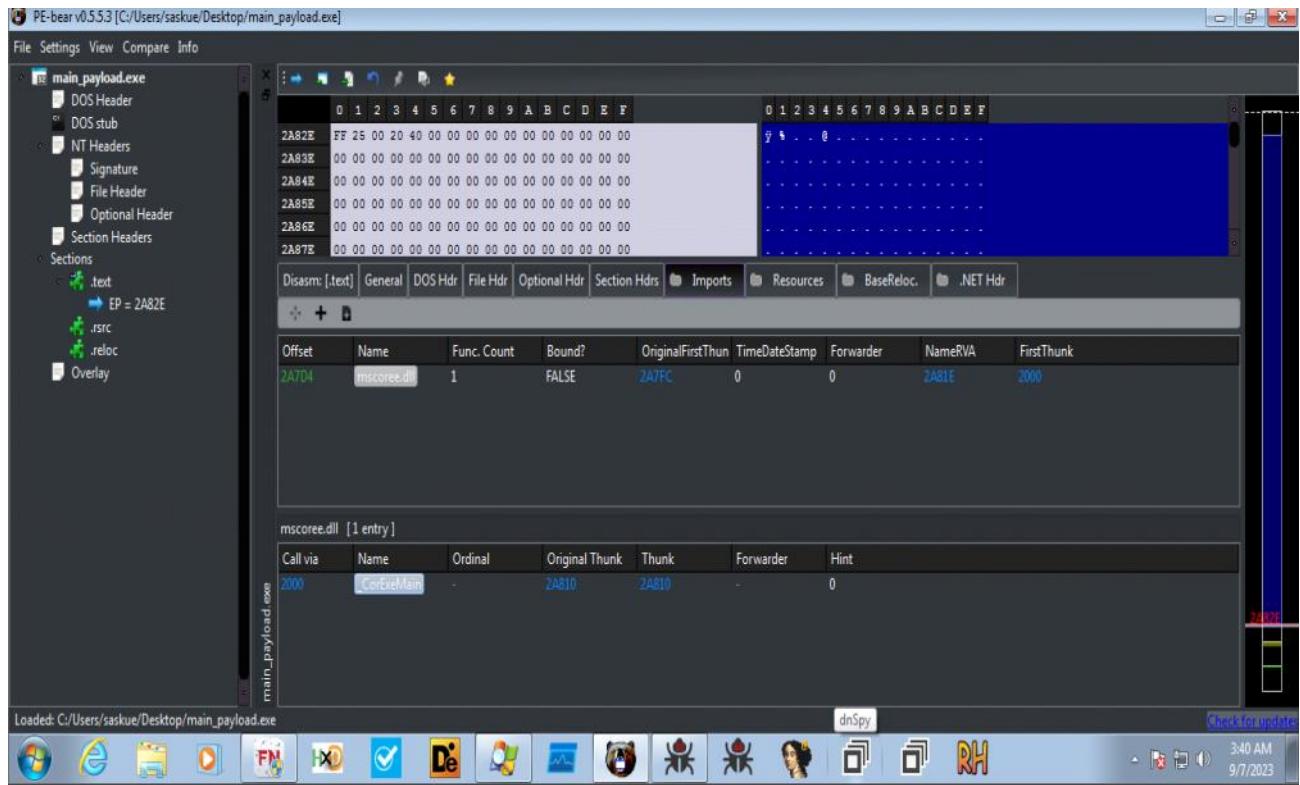
We will get pass the api and will see a PE Written to above mentioned memory region in the remote target process.

The figure displays two side-by-side debugger windows, likely from Immunity Debugger, showing the assembly and memory dump for a signed contract application. The left window shows assembly code for the main thread (Thread 1468) and the right window shows assembly code for a different thread (Thread 3992). Both windows have tabs for CPU, Log, Notes, Breakpoints, Memory Map, Call Stack, SEH, Script, and Symbols. The CPU tabs are active, showing assembly code. The left window's assembly shows instructions like mov, lea, and call, with registers EIP, ECX, and EAX being tracked. The right window's assembly shows instructions involving EIP, ECX, and EAX, with a specific instruction at address 77980184 jumping to address 779801B4. The memory dump tabs show hex and ASCII representations of memory, with the right window's dump showing a large block of binary data starting at address 00400000. The taskbar at the bottom shows various application icons, including the Windows Start button, a browser icon, and the debugger icons.

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

dnSpy v6.1.8 (32-bit, .NET)

File Edit View Debug Window Help C# Start

Assembly Explorer A C# X

```

25     [DllImport("user32.dll", EntryPoint = "GetKeyboardState")]
26     public static extern bool A(byte[]);
27
28     // Token: 0x05000012 RID: 18
29     [DllImport("user32.dll", EntryPoint = "MapVirtualKey")]
30     public static extern uint A(uint, uint);
31
32     // Token: 0x05000013 RID: 19
33     [DllImport("psapi.dll", EntryPoint = "EnumProcessModules")]
34     public static extern bool A(IntPtr, [MarshalAs(UnmanagedType.LPArray, ArraySubType = UnmanagedType.U4)] [In] [Out] uint[], uint,
35                               [MarshalAs(UnmanagedType.U4)] ref uint);
36
37     // Token: 0x05000014 RID: 20
38     [DllImport("psapi.dll", EntryPoint = "GetModuleFileNameEx")]
39     public static extern uint A(IntPtr, IntPtr, [Out] StringBuilder, [MarshalAs(UnmanagedType.U4)] [In] int);
40
41     // Token: 0x05000015 RID: 21
42     [DllImport("user32.dll", EntryPoint = "GetWindowThreadProcessId")]
43     public static extern int A(IntPtr, ref int);
44
45     // Token: 0x05000016 RID: 22
46     [DllImport("kernel32.dll", EntryPoint = "GetProcessId")]
47     public static extern int a(IntPtr);
48
49     // Token: 0x05000017 RID: 23
50     [DllImport("user32", EntryPoint = "GetKeyboardLayout")]
51     public static extern IntPtr A(int);
52
53     // Token: 0x05000018 RID: 24
54     [DllImport("user32", EntryPoint = "ToUnicodeEx")]
55     public static extern int A(uint, uint, byte[], IntPtr, [MarshalAs(UnmanagedType.LPStr)] [Out] StringBuilder, int, uint, IntPtr);

```

100 %

4:03 AM 9/7/2023

There is also dynamic loading of function from VaultCli.dll such as :

- 1- VaultOpenVault
- 2- VaultCloseVault
- 3- VaultFree
- 4- Others

dnSpy v6.1.8 (32-bit, .NET)

File Edit View Debug Window Help C# Start

Assembly Explorer A C# X

```

80
81     // Token: 0x0500001F RID: 31
82     [DllImport("vaultcli.dll", EntryPoint = "VaultOpenVault")]
83     public static extern int A(ref Guid, uint, ref IntPtr);
84
85     // Token: 0x05000020 RID: 32
86     [DllImport("vaultcli.dll", EntryPoint = "VaultCloseVault")]
87     public static extern int A(ref IntPtr);
88
89     // Token: 0x05000021 RID: 33
90     [DllImport("vaultcli.dll", EntryPoint = "VaultFree")]
91     public static extern int a(ref IntPtr);
92
93     // Token: 0x05000022 RID: 34
94     [DllImport("vaultcli.dll", EntryPoint = "VaultEnumerateVaults")]
95     public static extern int A(int, ref int, ref IntPtr);
96
97     // Token: 0x05000023 RID: 35
98     [DllImport("vaultcli.dll", EntryPoint = "VaultEnumerateItems")]
99     public static extern int A(IntPtr, int, ref int, ref IntPtr);
100
101    // Token: 0x05000024 RID: 36
102    [DllImport("vaultcli.dll", EntryPoint = "VaultGetItem")]
103    public static extern int A(IntPtr, ref Guid, IntPtr, IntPtr, IntPtr, int, ref IntPtr);
104
105    // Token: 0x05000025 RID: 37
106    [DllImport("vaultcli.dll", EntryPoint = "VaultGetItem")]
107    public static extern int A(IntPtr, ref Guid, IntPtr, IntPtr, IntPtr, int, ref IntPtr);
108
109    // Token: 0x05000026 RID: 38 RVA: 0x00002824 File Offset: 0x00002824
110    public static object B(IntPtr A 0);

```

100 %

7:43 PM 9/7/2023

Malware checking for tickcount

dnSpy v6.1.8 (32-bit, .NET)

File Edit View Debug Window Help C# Start

Assembly Explorer D X

```

34
35 // Token: 0x0600002B RID: 43 RVA: 0x00002F14 File Offset: 0x00002F14
36 public static uint A()
37 {
38     int num = 0;
39     do
40     {
41         if (num == 0)
42         {
43             num = 1;
44         }
45     }
46     while (num != 1);
47     uint result;
48     try
49     {
50         result = (uint)(Environment.TickCount - (int)D.a());
51     }
52     catch
53     {
54         result = 0U;
55     }
56     return result;
57 }
58
59 // Token: 0x0600002C RID: 44 RVA: 0x00002F60 File Offset: 0x00002F60
60 public static uint a()
61 {
62     int num = 0;
63     for (;;)
64     {
65         if (num == 1)
66     }
67 }

```

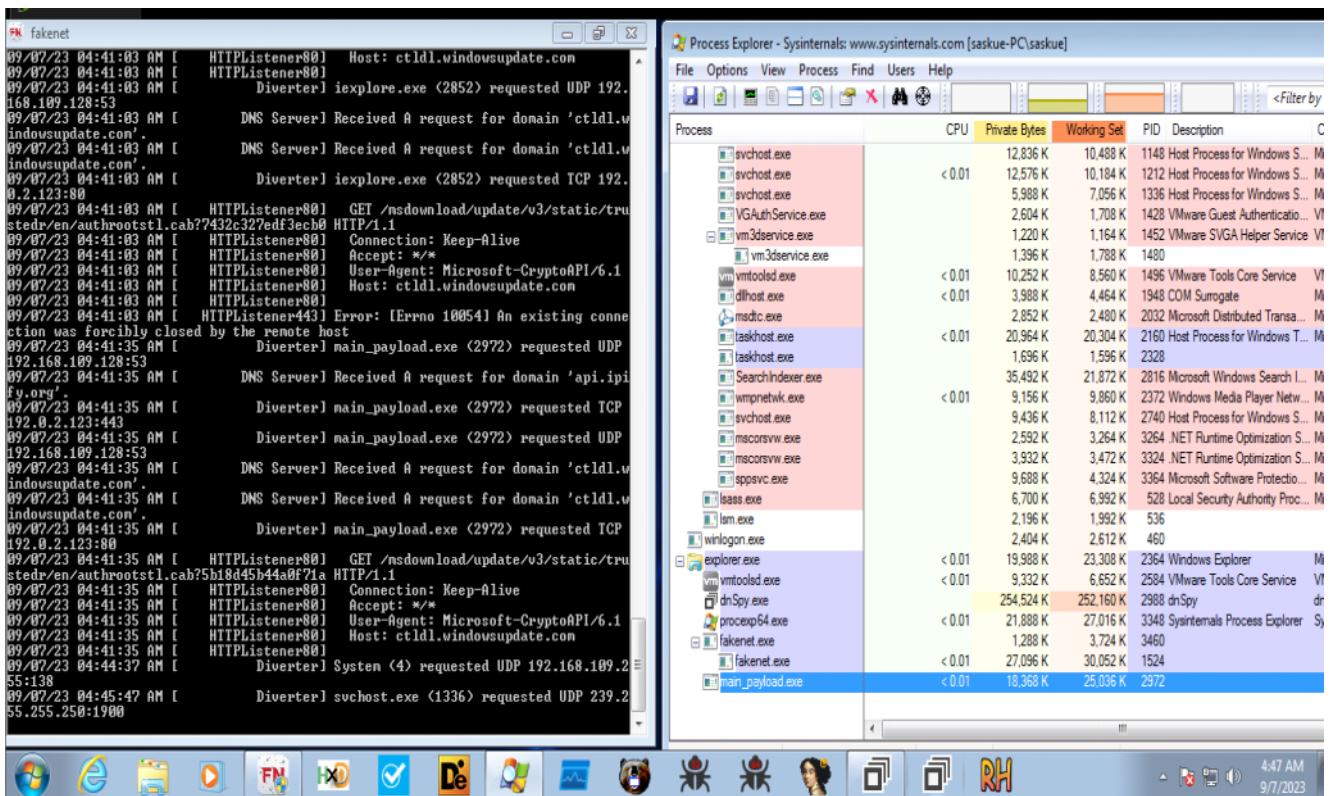
100 %

7:46 PM 9/7/2023

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

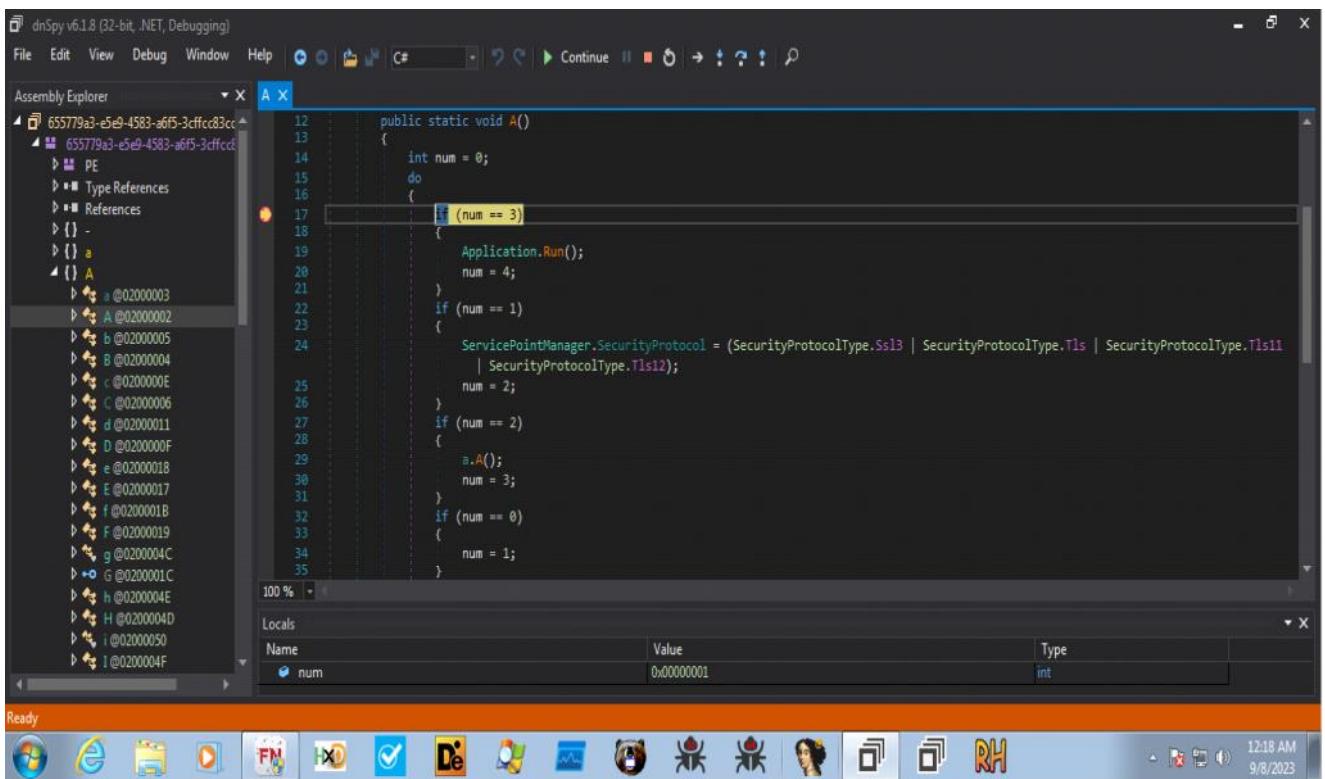


This was just a dry run test of malware. Let's get serious and start dissecting the malware.

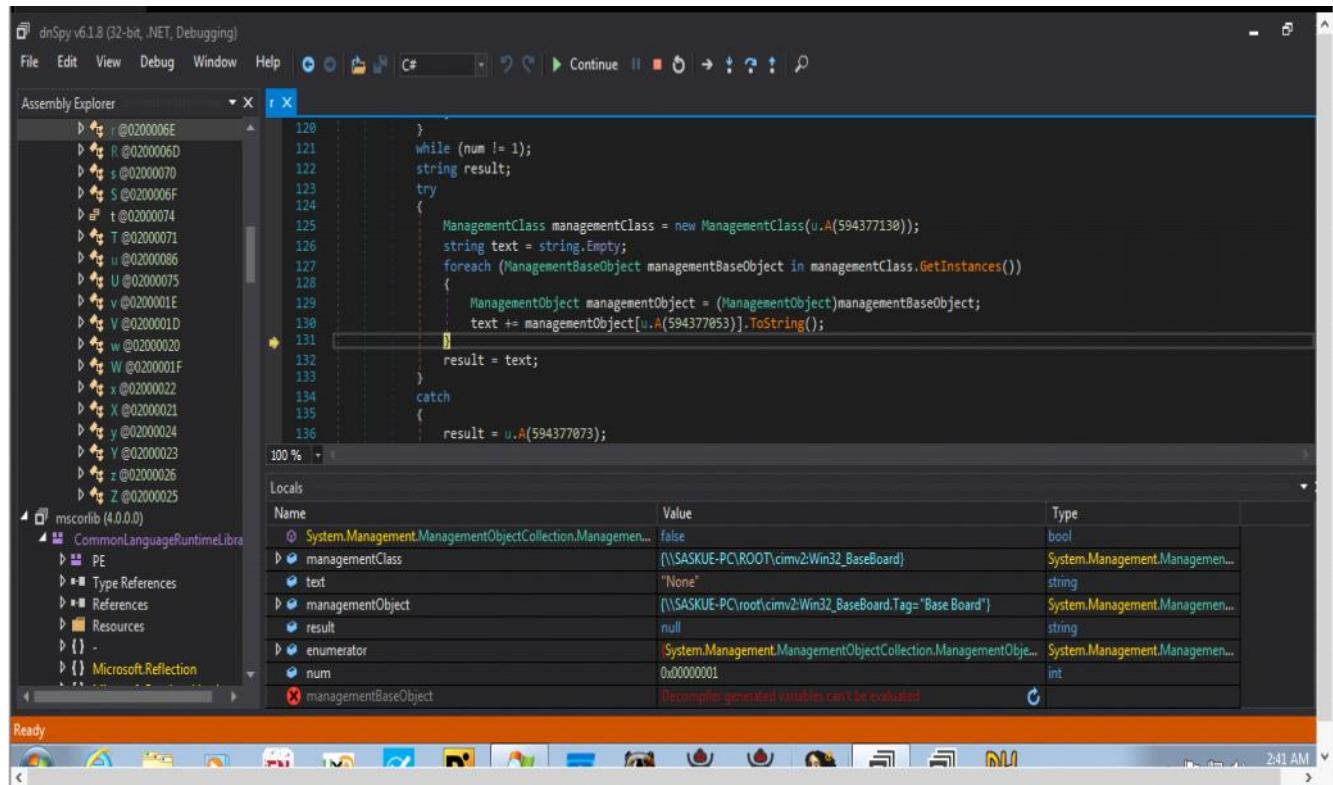
### Malware main capabilities:

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.



Assembly Explorer

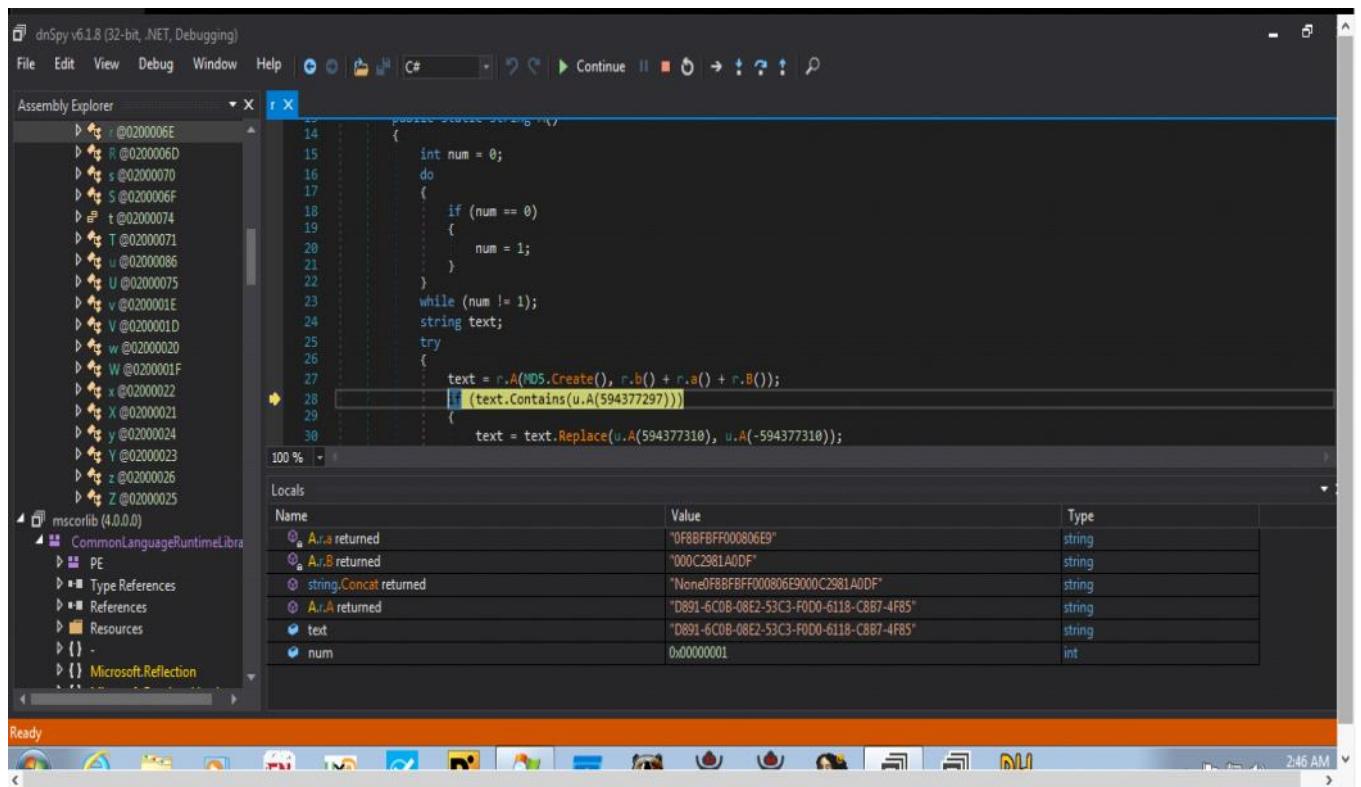
```

120     }
121     while (num != 1);
122     string result;
123     try
124     {
125         ManagementClass managementClass = new ManagementClass(u.A(594377130));
126         string text = string.Empty;
127         foreach (ManagementBaseObject managementBaseObject in managementClass.GetInstances())
128         {
129             ManagementObject managementObject = (ManagementObject)managementBaseObject;
130             text += managementObject[u.A(594377053)].ToString();
131         }
132         result = text;
133     }
134     catch
135     {
136         result = u.A(594377073);
137     }

```

Locals

Name	Type	Value
System.Management.ManagementObjectCollection.ManagementObjectCollection	System.Management.ManagementObjectCollection	false
managementClass	System.Management.ManagementClass	\\\SASKUE-PC\ROOT\cimv2\Win32_BaseBoard
text	string	"None"
managementObject	System.Management.ManagementObject	\\\SASKUE-PC\root\cimv2\Win32_BaseBoard.Tag="Base Board"
result	string	null
enumerator	System.Management.ManagementObjectCollection.ManagementObjectCollection	System.Management.ManagementObjectCollection.ManagementObjectCollection
num	int	0x00000001
managementBaseObject	System.Management.ManagementBaseObject	Decompiler generated variables can't be evaluated



Assembly Explorer

```

14     {
15         int num = 0;
16         do
17         {
18             if (num == 0)
19             {
20                 num = 1;
21             }
22         }
23         while (num != 1);
24         string text;
25         try
26         {
27             text = r.A(MDS.Create(), r.b() + r.a() + r.b());
28             if (text.Contains(u.A(594377297)))
29             {
30                 text = text.Replace(u.A(594377310), u.A(-594377310));
31             }
32         }
33     }

```

Locals

Name	Type	Value
A returned	string	"0F8BFBF000806E9"
A,B returned	string	"000C2981A0DF"
string.Concat returned	string	"None\0F8BFBF000806E9000C2981A0DF"
A,A returned	string	"D891-6C0B-08E2-53C3-F0D0-6118-C8B7-4F85"
text	string	"D891-6C0B-08E2-53C3-F0D0-6118-C8B7-4F85"
num	int	0x00000001

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

dnSpy v6.1.8 (32-bit, .NET, Debugging)

File Edit View Debug Window Help C# Continue

Assembly Explorer

```

1  using System;
2  using System.IO;
3  using System.Reflection;
4  using System.Windows.Forms;
5
6  namespace A
7  {
8      // Token: 0x0200005 RID: 5
9      public class b
10     {
11         // Token: 0x0600008 RID: 11 RVA: 0x0000284C File Offset: 0x0000284C
12         public static void A()
13         {
14             int num = 0;
15             do
16             {
17                 if (num == 2)

```

Locals

Name	Type
System.Reflection.Assembly.Location.get returned	string
num	int

Ready

Later on the malware get environment variable and concatenate it with other string which together look like this:

@ "C:\Users\saskue\AppData\Roaming\qXYojnj"

Assembly Explorer

```

1  using System;
2  using System.IO;
3  using System.Reflection;
4  using System.Windows.Forms;
5
6  namespace A
7  {
8      // Token: 0x0200005 RID: 5
9      public class b
10     {
11         // Token: 0x0600008 RID: 11 RVA: 0x0000284C File Offset: 0x0000284C
12         public static void A()
13         {
14             int num = 0;
15             do
16             {
17                 if (num == 2)

```

Locals

Name	Type
System.Environment.GetEnvironmentVariable returned	string
System.IO.Path.Combine returned	string
num	int

Ready

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.

```

20     num = 3;
21 }
22 if (num == 3)
23 {
24     global::A.b.n = Path.Combine(Environment.GetEnvironmentVariable(global::A.b.0), global::A.b.0);
25     num = 4;
26 }
27 if (num == 4)
28 {
29     global::A.b.N = Path.Combine(global::A.b.n, global::A.b.P);
30     num = 5;
31 }
32 if (num == 5)
33 {
34     global::A.b.b = SystemInformation.UserName + u.A(594361733) + SystemInformation.ComputerName;
35     num = 6;
36 }
37 if (num == 6)
38 {

```

Locals		
Name	Value	Type
System.Windows.Forms.SystemInformation.ComputerName.get..	"SASKUE-PC"	string
string.Concat returned	"saskue/SASKUE-PC"	string
num	0x00000005	int

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.

```

Assembly Explorer R x
117     {
118         num = 1;
119     }
120 }
121 while (num != 1);
122 string result;
123 try
124 {
125     HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(b.d);
126     httpWebRequest.Credentials = CredentialCache.DefaultCredentials;

```

fy.org' 09/08/23 03:19:57 AM [ Diverted] main\_payload.exe (3948) requested TCP 192.0.2.123:443
09/08/23 03:20:00 AM [ Diverted] main\_payload.exe (3948) requested UDP 192.168.109.128:53
09/08/23 03:20:00 AM [ DNS Server] Received A request for domain 'ctldl.windowsupdate.com'.
09/08/23 03:20:00 AM [ DNS Server] Received A request for domain 'ctldl.windowsupdate.com'.
09/08/23 03:20:00 AM [ Diverted] main\_payload.exe (3948) requested TCP 192.0.2.123:80
09/08/23 03:20:00 AM [ HTTPListener80] GET /msdownload/update/v3/static/trustedinstaller/en/authrootstl.cab?03c02071c0c81160 HTTP/1.1
09/08/23 03:20:00 AM [ HTTPListener80] Connection: Keep-Alive
09/08/23 03:20:00 AM [ HTTPListener80] Accept: \*/\*
09/08/23 03:20:00 AM [ HTTPListener80] User-Agent: Microsoft-CryptoAPI/6.1
09/08/23 03:20:00 AM [ HTTPListener80] Host: ctldl.windowsupdate.com
09/08/23 03:21:54 AM [ Diverted] System (4) requested UDP 192.168.109.255:137
09/08/23 03:22:09 AM [ Diverted] svchost.exe (1356) requested UDP 239.255.250:1900
09/08/23 03:22:34 AM [ Diverted] System (4) requested UDP 192.168.109.255:138

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

```

326     num = 1;
327     }
328   }
329   while (num != 1);
330   try
331   {
332     string processName = Process.GetCurrentProcess().ProcessName;
333     int id = Process.GetCurrentProcess().Id;
334     Process[] processesByName = Process.GetProcessesByName(processName);
335     foreach (Process process in processesByName)
336     {
337       if (process.Id != id)
338       {
339         process.Kill();
340       }
341     }
342   }
343   catch
344   {
345   }

```

Name	Value	Type
System.Diagnostics.Process.GetProcessesByName returned	[System.Diagnostics.Process[0x00000001]]	System.Diagnostics.Process[]
processName	"main_payload"	string
id	0x0000D70	int
processesByName	[System.Diagnostics.Process[0x00000001]]	System.Diagnostics.Process[]
[0]	{System.Diagnostics.Process (main_payload)}	System.Diagnostics.Process
process	null	System.Diagnostics.Process

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

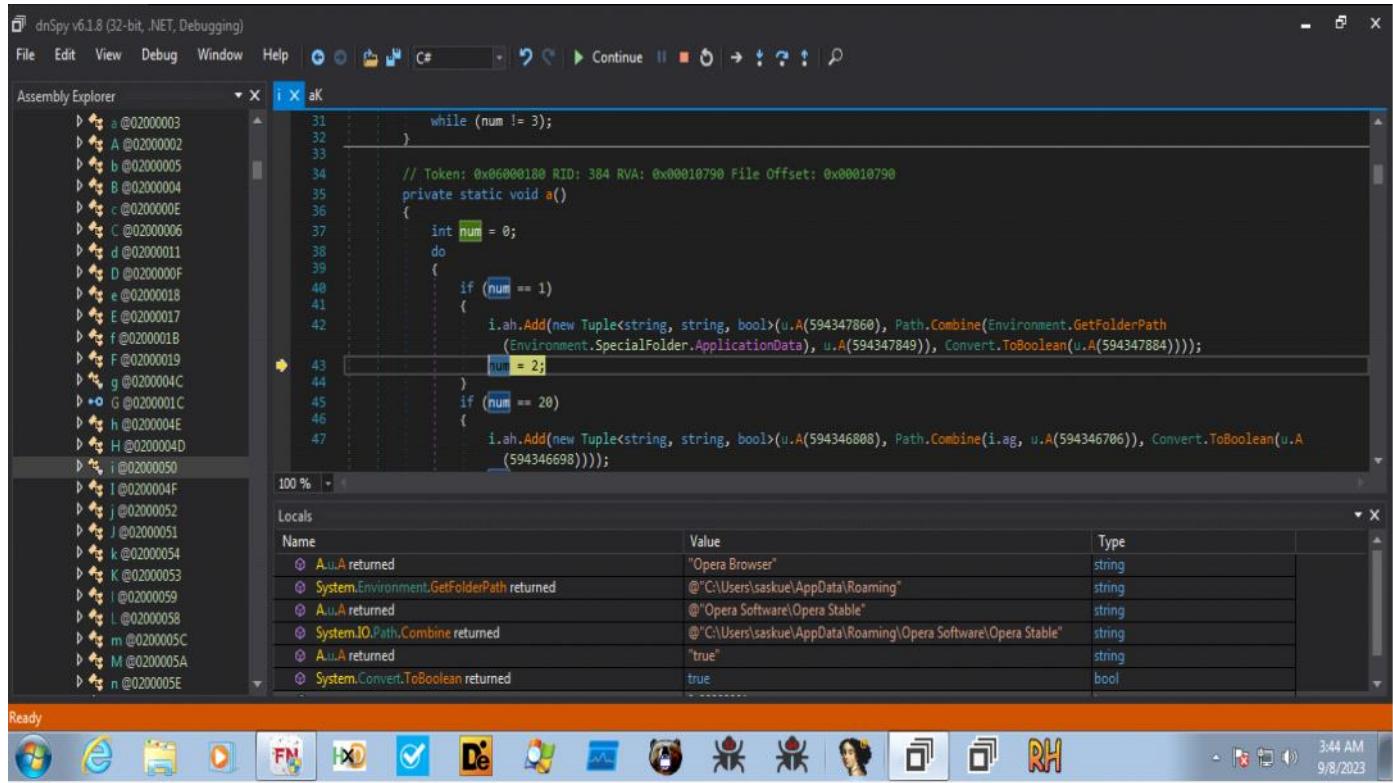
@ "C:\Users\saskue\AppData\Roaming\\$variable

Here \$variable is dynamic.

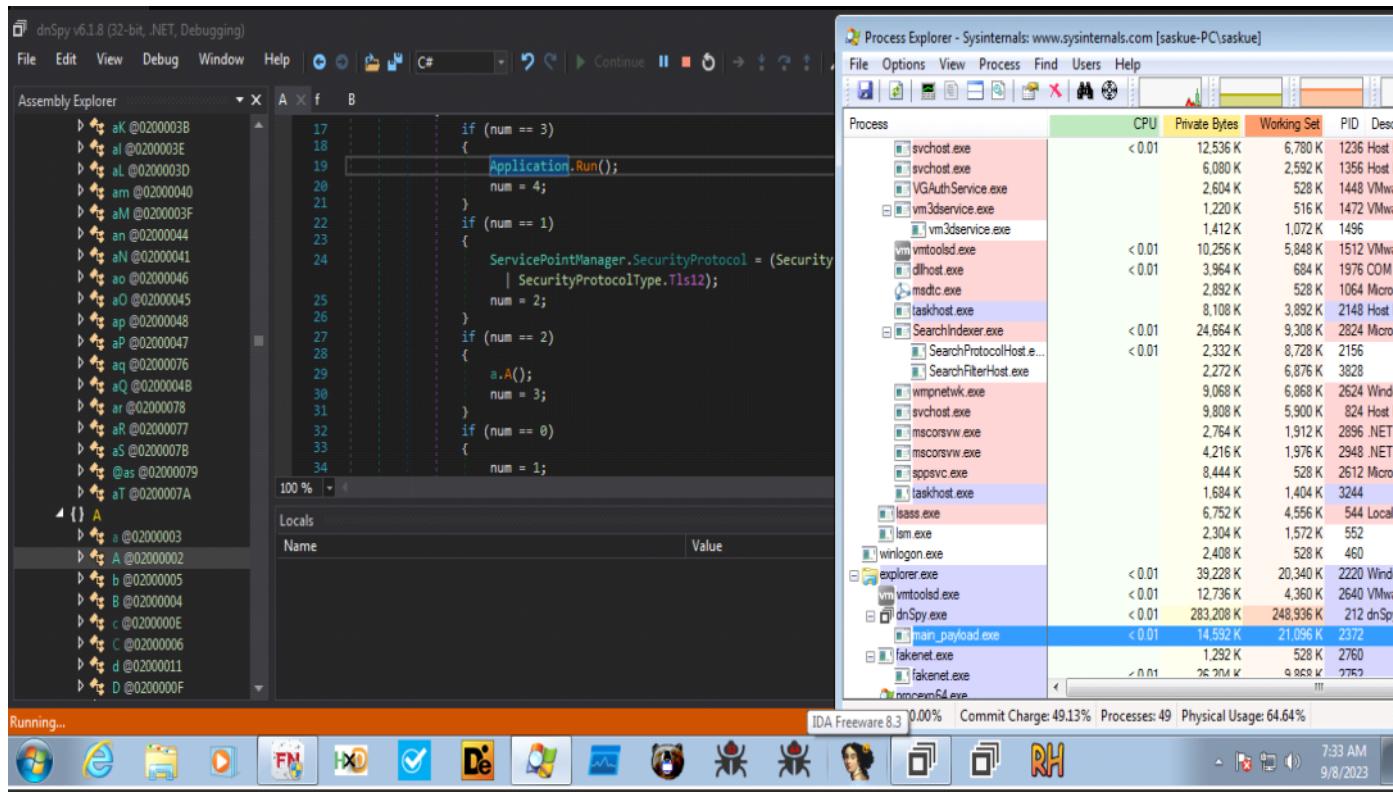
List of browser enumeration (dynamic part of path):

- 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
- ChangeClipboardChain
- SendMessage

```

4  using System.Windows.Forms;
5
6  namespace A
7  {
8      // Token: 0x02000019 RID: 25
9      public class F : NativeWindow
10     {
11         // Token: 0x0600005E RID: 94
12         [DllImport("user32", EntryPoint = "SetClipboardViewer")]
13         private static extern IntPtr A(IntPtr);
14
15         // Token: 0x0600005F RID: 95
16         [DllImport("user32", EntryPoint = "ChangeClipboardChain")]
17         private static extern bool A(IntPtr, IntPtr);
18
19         // Token: 0x06000060 RID: 96
20         [DllImport("user32", EntryPoint = "SendMessage")]
21         private static extern long A(IntPtr, int, IntPtr, IntPtr);
22
23         // Token: 0x06000061 RID: 97 RVA: 0x00004498 File Offset: 0x00004498
24         public void A(F, A a)
    }

```

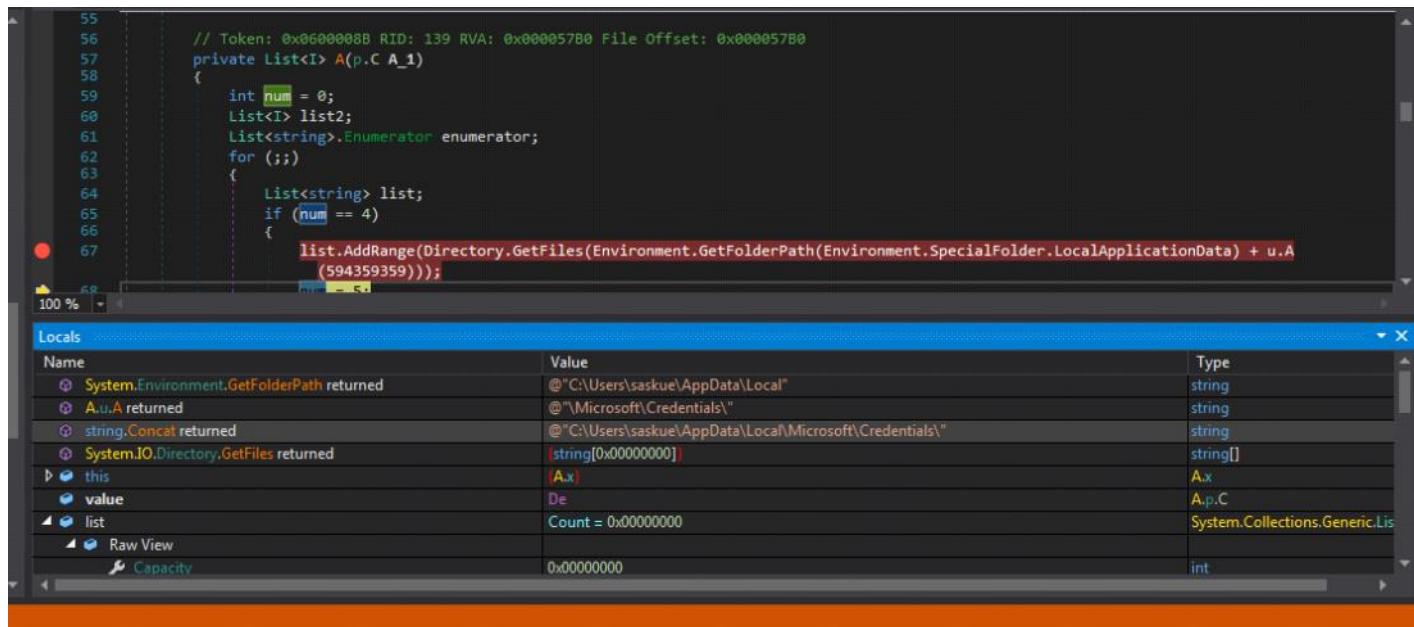
I could also see some cryptographic function too such as bcryptGetProperty etc.

```

2  using System.Runtime.InteropServices;
3
4  namespace A
5  {
6      // Token: 0x02000063 RID: 99
7      public static class p
8      {
9          // Token: 0x060000E3 RID: 483
10         [DllImport("bcrypt.dll", EntryPoint = "BCryptOpenAlgorithmProvider")]
11         public static extern uint A(out IntPtr, [MarshalAs(UnmanagedType.LPWSTR)] string, [MarshalAs(UnmanagedType.UnmanagedType.LPWSTR)] string, uint);
12
13         // Token: 0x060000E4 RID: 484
14         [DllImport("bcrypt.dll", EntryPoint = "BCryptCloseAlgorithmProvider")]
15         public static extern uint A(IntPtr, uint);
16
17         // Token: 0x060000E5 RID: 485
18         [DllImport("bcrypt.dll", EntryPoint = "BCryptGetProperty")]
19         public static extern uint A(IntPtr, [MarshalAs(UnmanagedType.LPWSTR)] string, byte[], int, ref int, uint);
20

```

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.



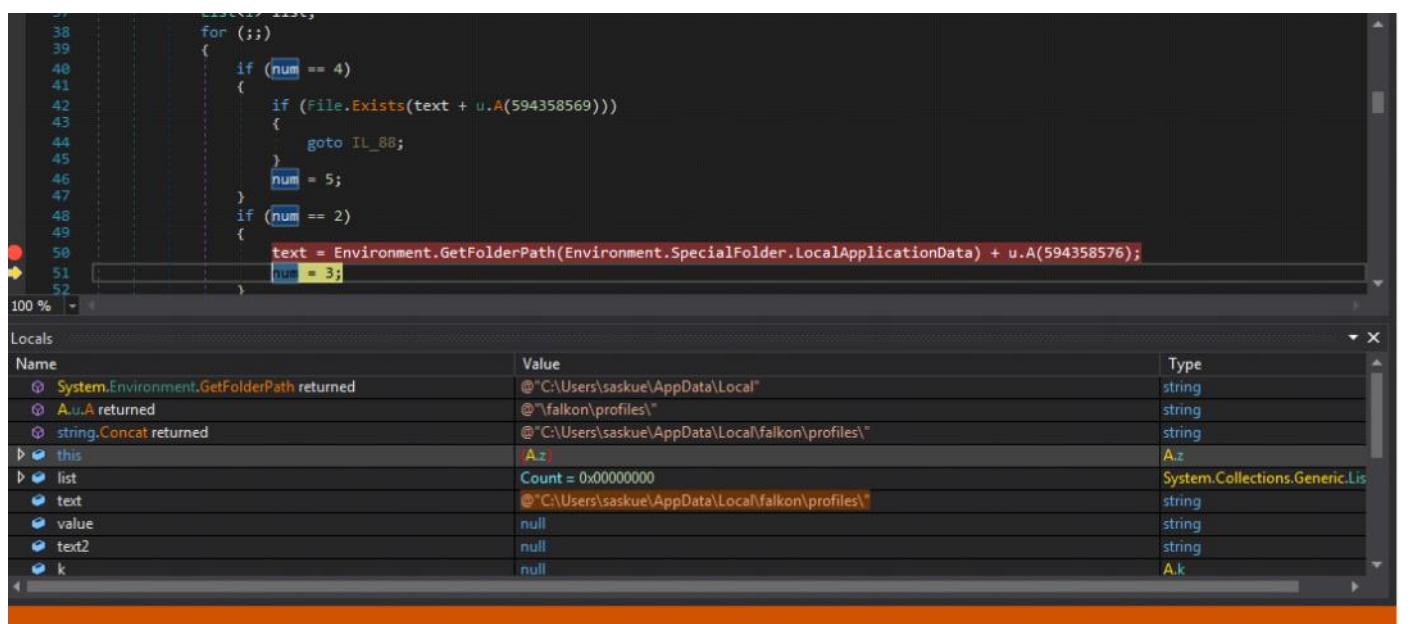
```

55
56     // Token: 0x0600008B RID: 139 RVA: 0x000057B0 File Offset: 0x000057B0
57     private List<I> A(p.C A_1)
58     {
59         int num = 0;
60         List<I> list2;
61         List<string>.Enumerator enumerator;
62         for (;;)
63         {
64             List<string> list;
65             if (num == 4)
66             {
67                 list.AddRange(Directory.GetFiles(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + u.A(594259359)));

```

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

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



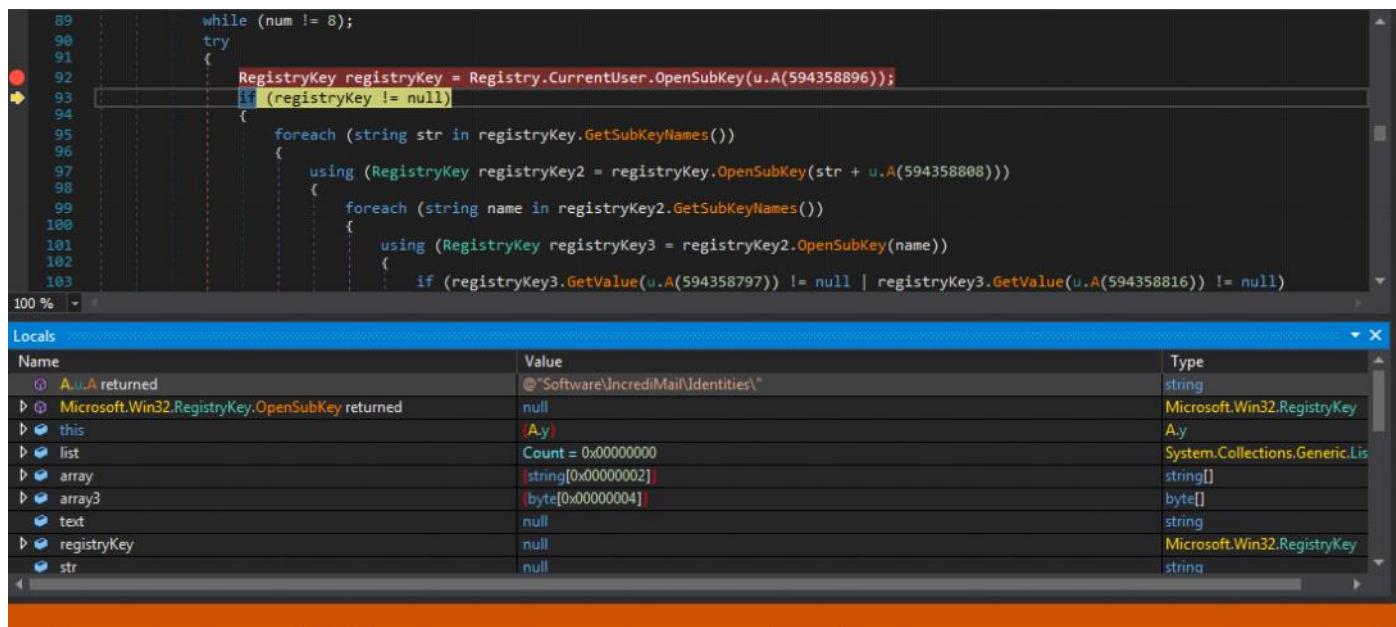
```

37         list2 = list;
38         for (;;)
39         {
40             if (num == 4)
41             {
42                 if (File.Exists(text + u.A(594358569)))
43                 {
44                     goto IL_88;
45                 }
46                 num = 5;
47             }
48             if (num == 2)
49             {
50                 text = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + u.A(594358576);
51             }
52         }

```

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

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



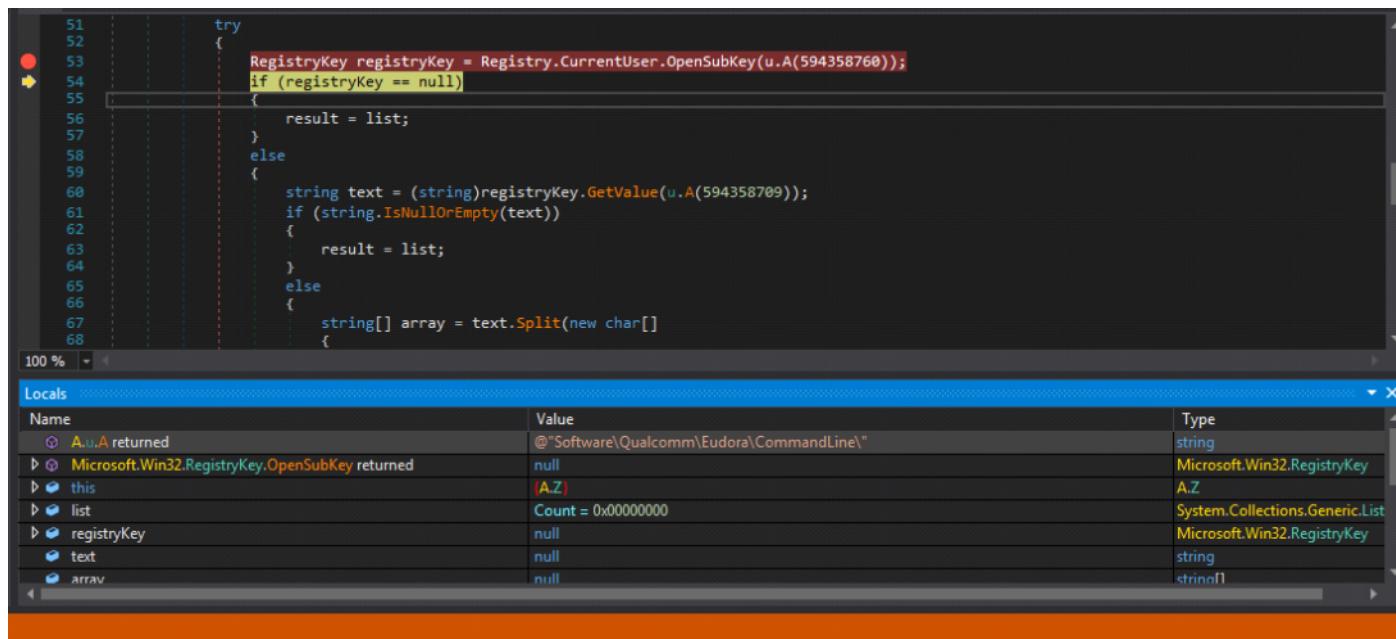
The screenshot shows a debugger interface with assembly code and a local variables table. The assembly code is as follows:

```
89     while (num != 8);
90     try
91     {
92         RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(u.A(594358896));
93         if (registryKey != null)
94         {
95             foreach (string str in registryKey.GetSubKeyNames())
96             {
97                 using (RegistryKey registryKey2 = registryKey.OpenSubKey(str + u.A(594358808)))
98                 {
99                     foreach (string name in registryKey2.GetSubKeyNames())
100                     {
101                         using (RegistryKey registryKey3 = registryKey2.OpenSubKey(name))
102                         {
103                             if (registryKey3.GetValue(u.A(594358797)) != null | registryKey3.GetValue(u.A(594358816)) != null)
```

The local variables table (labeled 'Locals') is as follows:

Name	Type	Value
u.A returned	string	@"Software\Incredimail\Identities\"
Microsoft.Win32.RegistryKey.OpenSubKey returned	Microsoft.Win32.RegistryKey	null
this	A.y	(A.y)
list	System.Collections.Generic.List	Count = 0x00000000
array	string[]	[string[0x00000002]]
array3	byte[]	[byte[0x00000004]]
text	string	null
registryKey	Microsoft.Win32.RegistryKey	null
str	string	null

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.



The screenshot shows a debugger interface with assembly code and a local variables table. The assembly code is as follows:

```
51     try
52     {
53         RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(u.A(594358760));
54         if (registryKey == null)
55         {
56             result = list;
57         }
58         else
59         {
60             string text = (string)registryKey.GetValue(u.A(594358709));
61             if (string.IsNullOrEmpty(text))
62             {
63                 result = list;
64             }
65             else
66             {
67                 string[] array = text.Split(new char[]
68                 {
```

The local variables table (labeled 'Locals') is as follows:

Name	Type	Value
u.A returned	string	@"Software\Qualcomm\Eudora\CommandLine\"
Microsoft.Win32.RegistryKey.OpenSubKey returned	Microsoft.Win32.RegistryKey	null
this	A.Z	(A.Z)
list	System.Collections.Generic.List	Count = 0x00000000
registryKey	Microsoft.Win32.RegistryKey	null
text	string	null
array	string[]	null

There is also enumeration of system information such as:

- 1- OS Version
- 2- System GUID

```
48         if (num == 3)
49         {
50             List<I> result;
51             return result;
52         }
53     }
54     try
55     {
56         Version version = Environment.OSVersion.Version;
57         int major = version.Major;
58         int minor = version.Minor;
59         Type typeFromHandle;
60         if (major >= 6 && minor >= 2)
61         {
62             typeFromHandle = typeof(C);
63         }
64     }
65 }
```

0 %

calls

ame	Value	Type
System.Environment.OSVersion.get returned	{Microsoft Windows NT 6.1.7601 Service Pack 1}	System.OperatingSystem
System.OperatingSystem.Version.get returned	(6.1.7601.65536)	System.Version
this	(A.W)	A.W
list	Count = 0x00000000	System.Collections.Generic.List
version	(6.1.7601.65536)	System.Version
major	0x00000000	int
minor	0x00000000	int

```
101
102
103     new Guid(u.A(594360091),
104     u.A(594360103)
105   },
106   {
107     new Guid(u.A(594360013),
108     u.A(594359961)
109   },
110   {
111     new Guid(u.A(594359996)),
```

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



8a0cfffb250244f00f8e61597d33ba01be11300b420de556cac6768f602ff00



Sign in

Sign up

Community Score: 27 / 65

① 27 security vendors and 1 sandbox flagged this file as malicious

8a0cfffb250244f00f8e61597d33ba01be11300b420de556cac6768f602ff00

Signed Contract.zip

Size: 771.66 KB | Last Analysis Date: 2 months ago | ZIP

Community Score: 27 / 65

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 1

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label: ① trojan.msil/noon

Threat categories: trojan

Family labels: msil noon pwsx

Security vendors' analysis ①

Security Vendor	Detected Threat	Confidence	Do you want to automate checks?
Avast	① Win32:PWSX-gen [Trj]	AVG	① Win32:PWSX-gen [Trj]
Cyren	① ZIP/Bredolab.AICamelot	DeepInstinct	① MALICIOUS
DrWeb	① Trojan.PWS.Siggen3.30229	Elastic	① Malicious (high Confidence)
ESET-NOD32	① A Variant Of MSIL/GenKryptik.GLAY	Fortinet	① MSIL/Kryptik.AGUHltr

Do you want to automate checks?

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 \(github.com\)](https://github.com/Deepanjalkumar/Malware-Signature/blob/main/Malware-Signature/Agent%20Tesla/agentesla.yar)

```
agentesla.yar
1 rule agent_tesla
2 {
3     meta:
4         author = "Deepanjali Kumar"
5         description="Agent tesla is a malware which is a stealer capable of stealing information such as microsoft cred
6         | | | keystrokes as well apart from stealing browser creds and email sensitive info"
7         org ="Operation Falcon"
8         date="9/9/2023"
9         md5="A8D2EBA5D6F81FFA035C5B7633C3D543"
10        sha256="7F52A1B5ED77D701595F1DE3543C7C4567B521EFCAF5EA12EEA93F55BA1203D1"
11
12    strings:
13        $binary="655779a3-e5e9-4583-a6f5-3cffcc83cda2.exe"
14        $privatedetails="<PrivateImplementationDetails>{3F199F7E-B77D-4E68-8CDF-CDCEEA1BE4D1}"
15        $strangestring="$2785b0b4-42ac-406a-b6aa-e12e31686fb4"
16        $strangestringnew="69754B7763534E"
17        $emailaddr="tranquochung252001@gmail.com"
18
19    condition:
20        any of them
21
22 }
```

#### SIGMA RULES:

[Malware-Signature/Agent Tesla/agentesla.yml at main · Deepanjalkumar/Malware-Signature \(github.com\)](https://github.com/Deepanjalkumar/Malware-Signature/blob/main/Malware-Signature/Agent%20Tesla/agentesla.yml)

```

agentesla.yml
1  title: agent tesla
2  ruletype: Sigma
3  author: Deepanjal kumar
4  date: 2023/09/09
5  description: Agent tesla is a malware which is a stealer capable of stealing information such as microsoft credentials
6  ||||| keystrokes as well apart from stealing browser creds and email sensitive info
7  detection:
8    SELECTION_1:
9      EventID: 4688
10   SELECTION_2:
11     Channel: Security
12   SELECTION_3:
13     Hashes:
14     - '*MD5=A8D2EBA5D6F81FFA035C5B7633C3D543*'
15     - '*SHA256=7F52A1B5ED77D701595F1DE3543C7C4567B521EFCAF5EA12EEA93F55BA1203D1*'
16     - '*IMPHASH=F190ED098182C8C4D9A5B5E866BC9CED*'
17   SELECTION_4:
18     md5: 7B9B91CAABD2288177ADB28DFC05259E
19   SELECTION_5:
20     sha256: 6E86F0BBACD63A35C78B533CC2C223374D5E2CA9938169A6C7A312211494F728
21   SELECTION_6:
22     Imphash: 2916DDA3C80B39A540B60C072A91A915
23   condition: ((SELECTION_1 and SELECTION_2) and ((SELECTION_3 or
24     (SELECTION_4 or SELECTION_5 or SELECTION_6)))
25   falsepositives:
26   - 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