Hunting AsyncRAT & QuasarRAT

dfir.ch/posts/asyncrat_quasarrat/

January 15, 2024

15 Jan 2024

Introduction

Recorded Future writes in their Adversary Infrastructure Report 2023:

The top 5 malware families we detected this year are AsyncRAT, Quasar RAT, PlugX, ShadowPad, and DarkComet. Interestingly, the top 2 detections are open-source, and the last 3 are well-established tools, showing that our statement from last year's report remains true:

[The] high level of commodity tool use indicates that threat actors are more concerned with blending in and being non-attributable rather than being undetectable, or have simply determined that their targets are not likely to detect even these well-known tools.

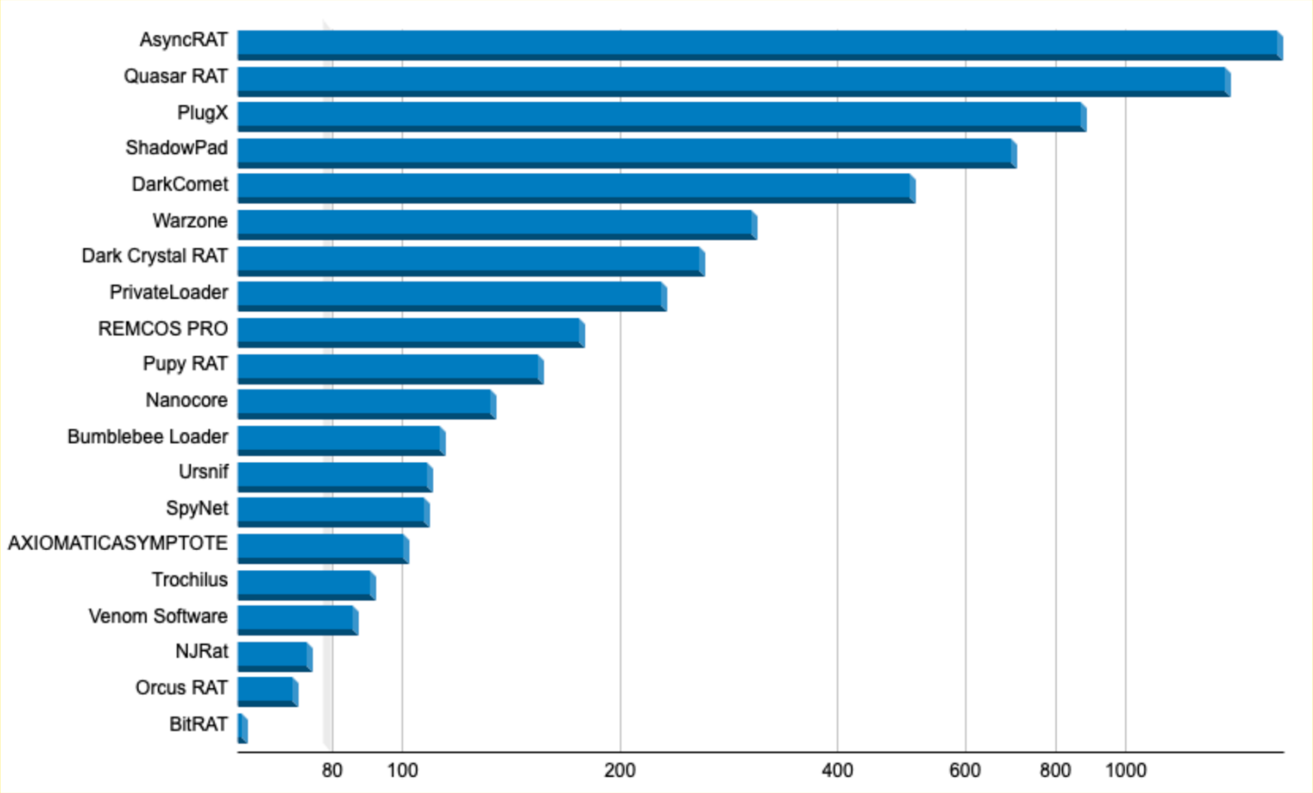


Figure 1: Top 20 Remote Access Trojans (RATs) and backdoors, ranked according to the number of unique Command and Control (C2) servers observed. (Source: Recorded Future)

The final statement in the aforementioned quote is particularly interesting: *[…] have simply determined that their targets are not likely to detect even these well-known tools.* This post combines and extends two longer threads previously shared on X (formerly Twitter) by me, discussing AsyncRAT and QuasarRAT. It highlights techniques and indicators of compromise (IOCs) that we, as defenders, can leverage to identify and hunt infections in our environment caused by these two types of remote access trojans.

AsyncRAT

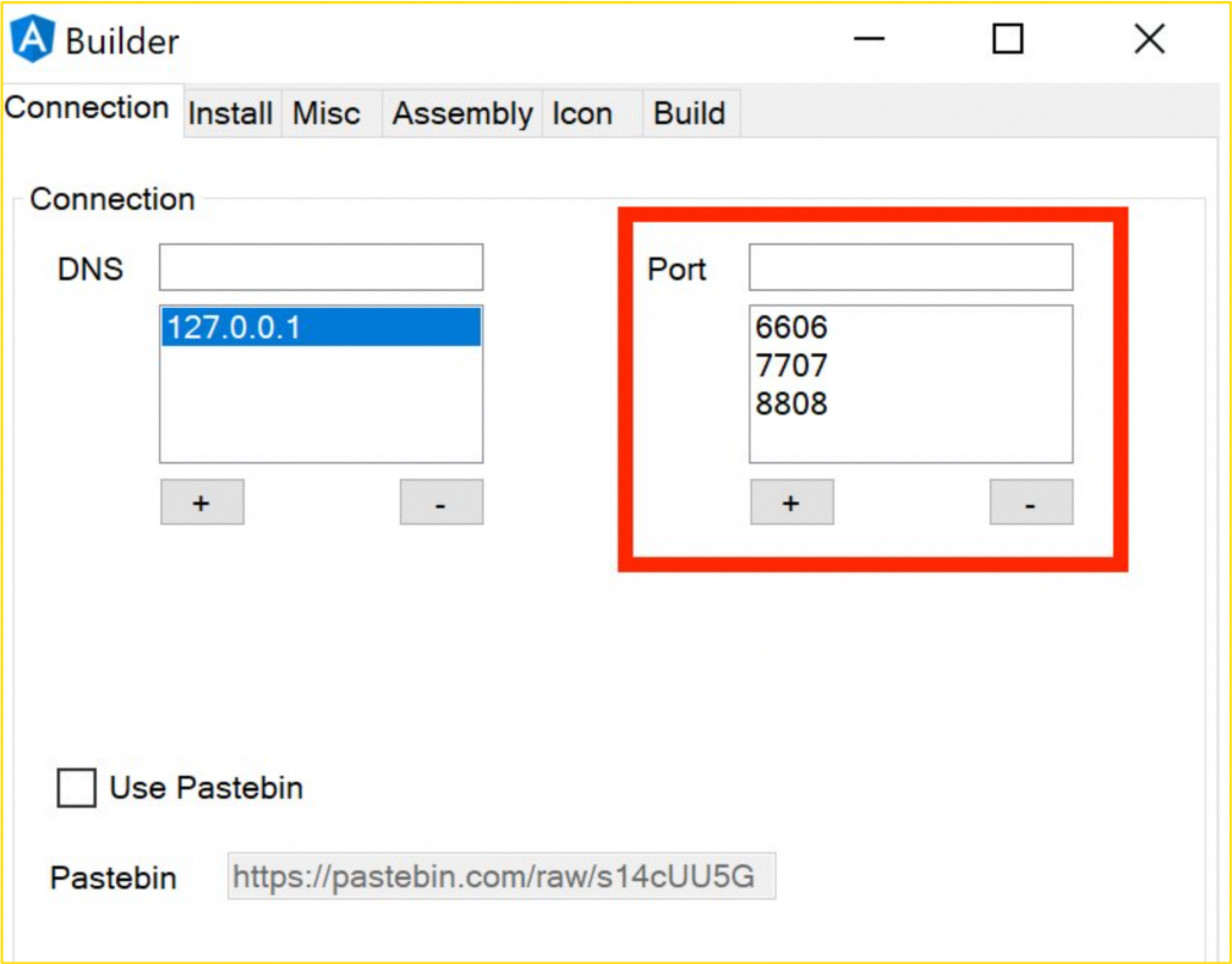
AsyncRAT is a prevalent Trojan executed at the end of a (potentially longer) infection chain on target computers. HP Security and Trellix have recently reported that attackers have been deploying AsyncRAT. Since the source code of AsyncRAT is publicly available, we obtained a copy of the builder to scrutinize its features and build detection capabilities for this Remote Access Trojan (RAT).

Several methods can be employed to identify endpoints infected with AsyncRAT:

- Detection of standard Command and Control (C2) ports usage.
- Search for persistence mechanisms.
- Examination of Mutexes.
- Last but not least, hunting for dropped DLL files.

Standard Command and Control (C2) Ports

The builder shows three default ports (6606, 7707, 8808) when assembling a new AsyncRAT client. Within the same interface, users can input the C2 address, to which the AsyncRAT client will establish a connection upon successful infection. Additionally, there is an option to store the C2 address on Pastebin.



Connection

DNS
127.0.0.1

Port
6606
7707
8808

□ Use Pastebin

Pastebin https://pastebin.com/raw/s14cUU5G

Figure 2: AsyncRat's default ports

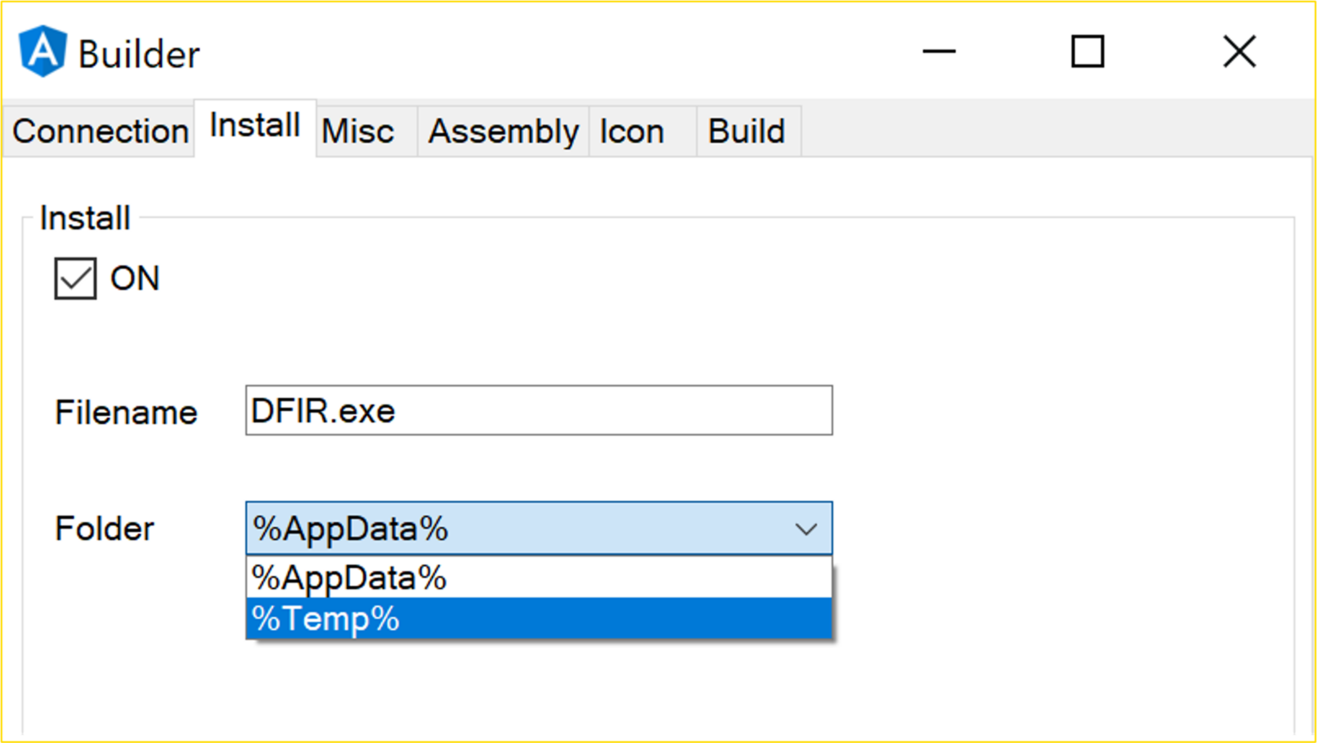
Analysis of 1000 AsyncRAT samples from ThreatFox revealed that, as of January 2024, the top three ports for the Command and Control (C2) address were consistently 6606, 7707, and 8808. This indicates that many attackers did not take the effort to modify the default port within the builder. Additionally, it was observed that all samples accessible on Bazaar utilized a C2 port higher than 1024. This information might be helpful for hunts within the Firewall logs (search especially for connections to IP-only domains connecting to a high-port - but beware: this search might return many hits and, without the context of the process that created the network connection, a daunting task).

To retrieve Indicators of Compromise (IOCs) from AsyncRAT samples on ThreatFox, we used the following curl command:

```
curl -X POST https://threatfox-api.abuse.ch/api/v1/ -d '{ "query": "taginfo", "tag":
"AsyncRAT", "limit": 1000 }'
```

Persistence

In the builder, you can specify whether a persistence should be set up or not. When this option is selected, you can define a filename and choose the directory where the payload will be stored (per default, only two directories are available). However, as the source code is available, expanding the selection to include additional paths should be a straightforward task.



Install

☑ ON

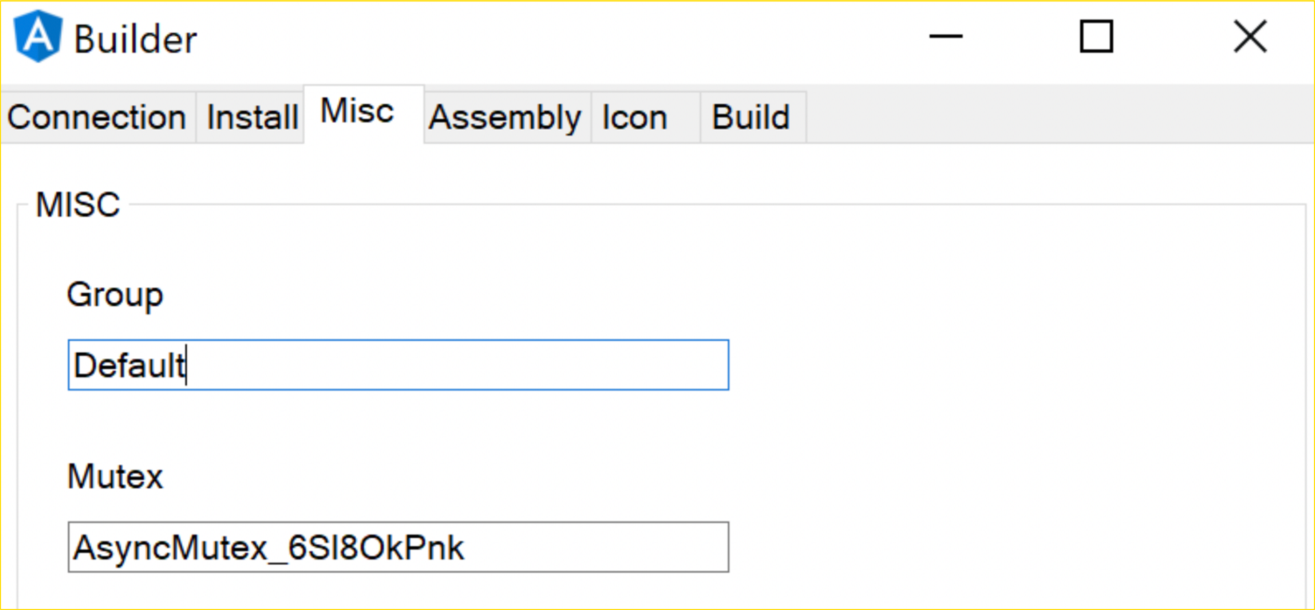
Filename DFIR.exe

Folder %AppData%

Figure 3: Installation directories

Mutex

AsyncRAT uses a mutex to detect an already infected system. The prefix of the mutex “AsyncMutex_” was also mentioned in HP Security’s tweet, indicating that attackers use the default settings in some cases (which is good for us defenders; read the section about Mutex hunting below).



MISC

Group

Default

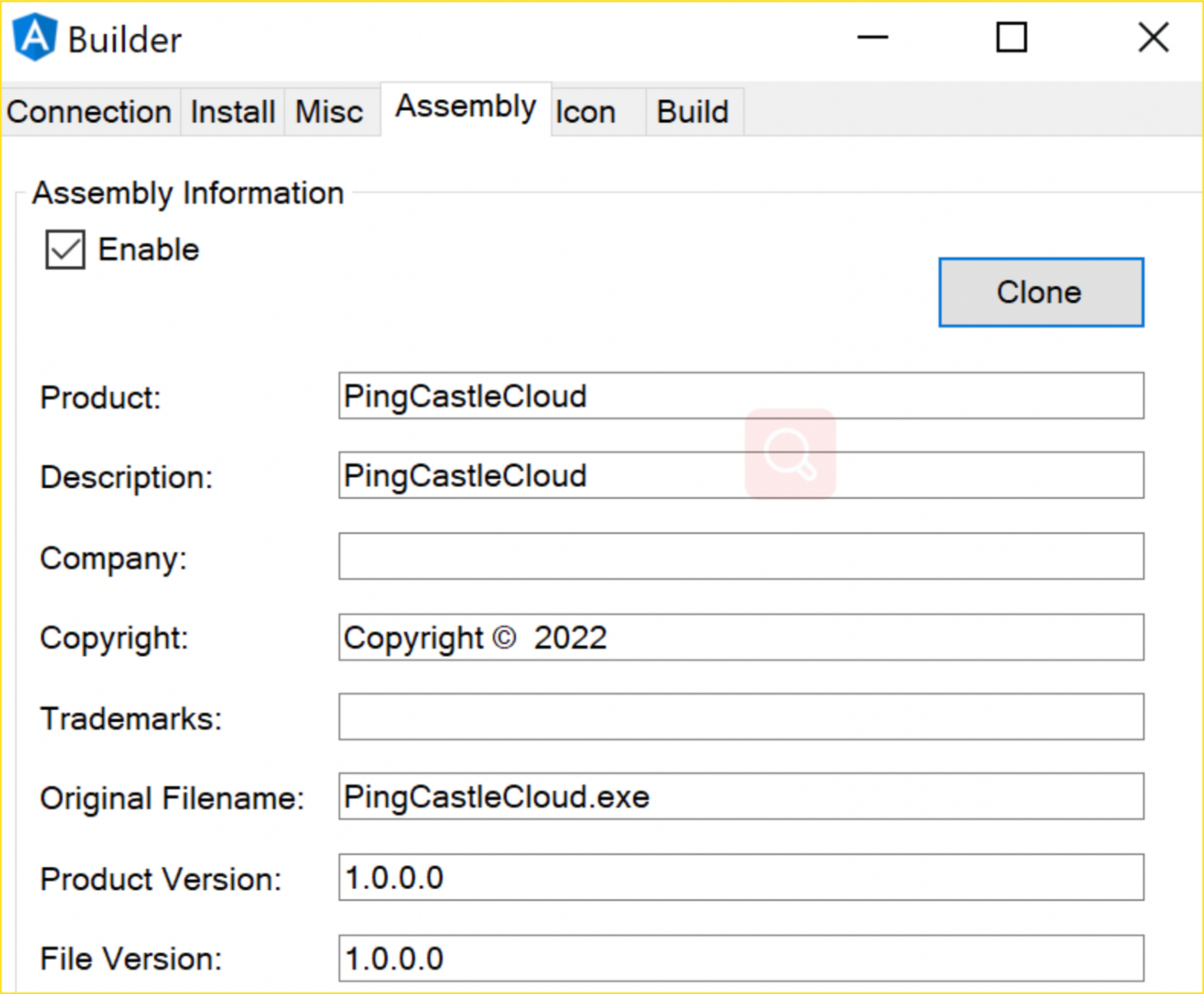
Mutex

AsyncMutex_6SI8OkPnk

Figure 4: Default Mutex

Assembly Information

Interestingly, we can “clone” the information about the binary from another binary for blending in. In this example, I took the data from the legitimate PingCastleCloud binary - the builder then used the same information for the malicious binary.



Assembly Information

☑ Enable

Product: PingCastleCloud

Description: PingCastleCloud

Company:

Copyright: Copyright © 2022

Trademarks:

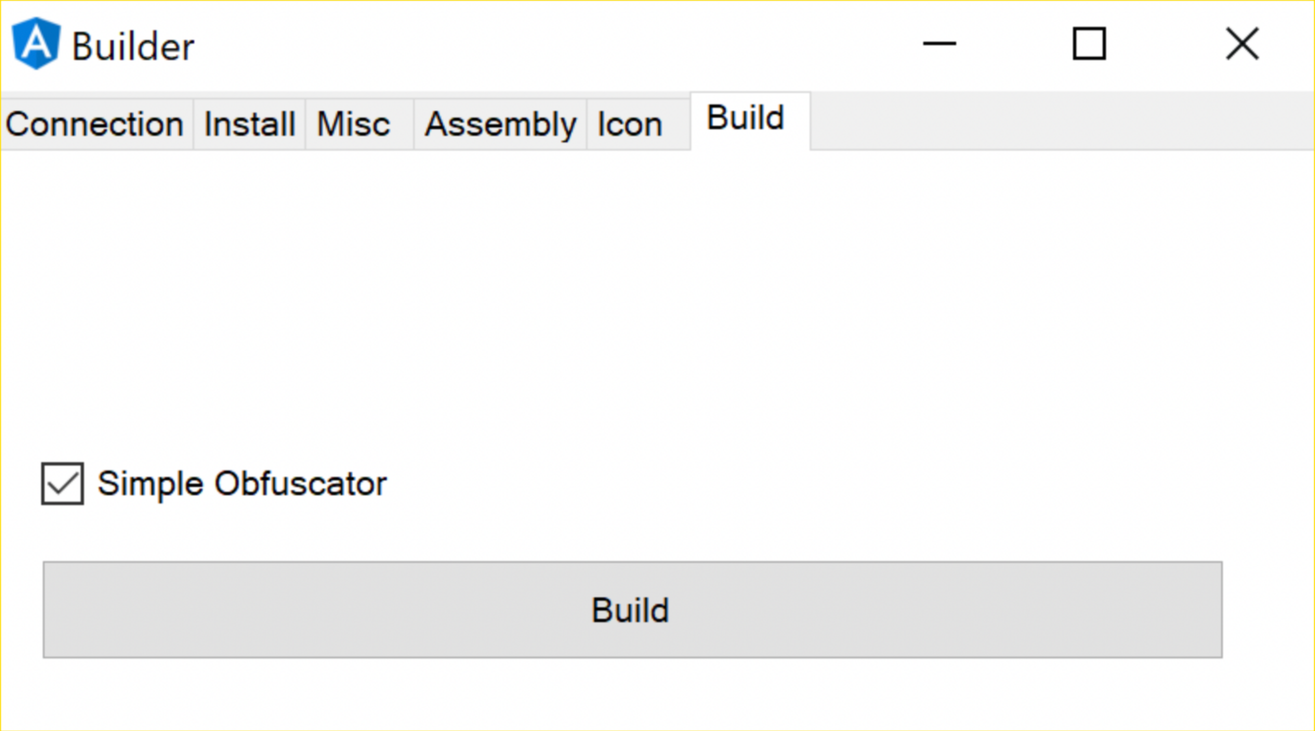
Original Filename: PingCastleCloud.exe

Product Version: 1.0.0.0

File Version: 1.0.0.0

Figure 5: Tamper the assembly information

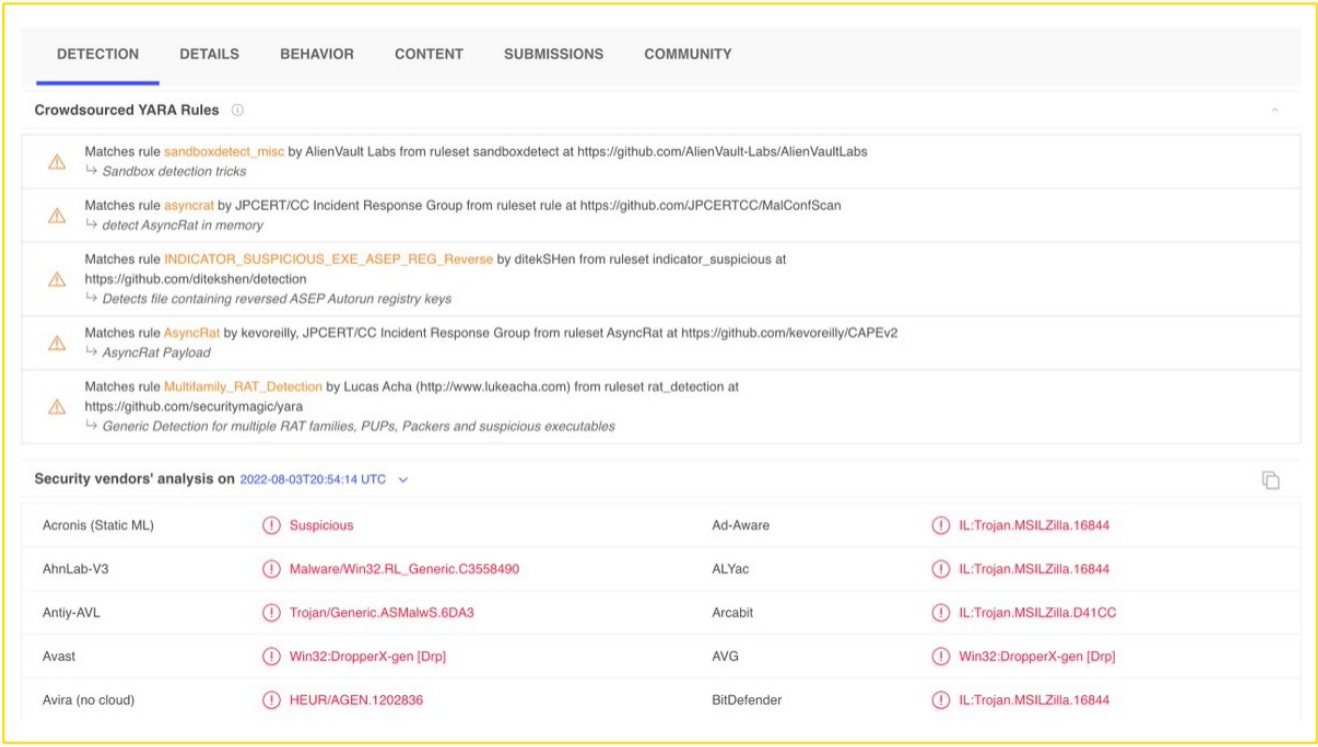
We can also force an obfuscation of the .NET code to slow down Reverse Engineering (I haven’t looked into this feature).



☑ Simple Obfuscator

Figure 6: Simple obfuscation mechanism

The sample generated through the outlined steps above comes straight to a VT score of 48 (as of the time when I initially posted the original tweets) - anything but unknown, also detected by various YARA signatures.



Crowdsourced YARA Rules

⚠ Matches rule sandboxdetect_misc by AlienVault Labs from ruleset sandboxdetect at https://github.com/AlienVault-Labs/AlienVaultLabs
↳ *Sandbox detection tricks*

⚠ Matches rule asyncrat by JPCERT/CC Incident Response Group from ruleset rule at https://github.com/JPCERTCC/MalConfScan
↳ *detect AsyncRat in memory*

⚠ Matches rule INDICATOR_SUSPICIOUS_EXE_ASEP_REG_Reverse by ditekSHen from ruleset indicator_suspicious at https://github.com/ditekshen/detection
↳ *Detects file containing reversed ASEP Autorun registry keys*

⚠ Matches rule AsyncRat by kevoreilly, JPCERT/CC Incident Response Group from ruleset AsyncRat at https://github.com/kevoreilly/CAPEv2
↳ *AsyncRat Payload*

⚠ Matches rule Multifamily_RAT_Detection by Lucas Acha (http://www.lukeacha.com) from ruleset rat_detection at https://github.com/securitymagic/yara
↳ *Generic Detection for multiple RAT families, PUPs, Packers and suspicious executables*

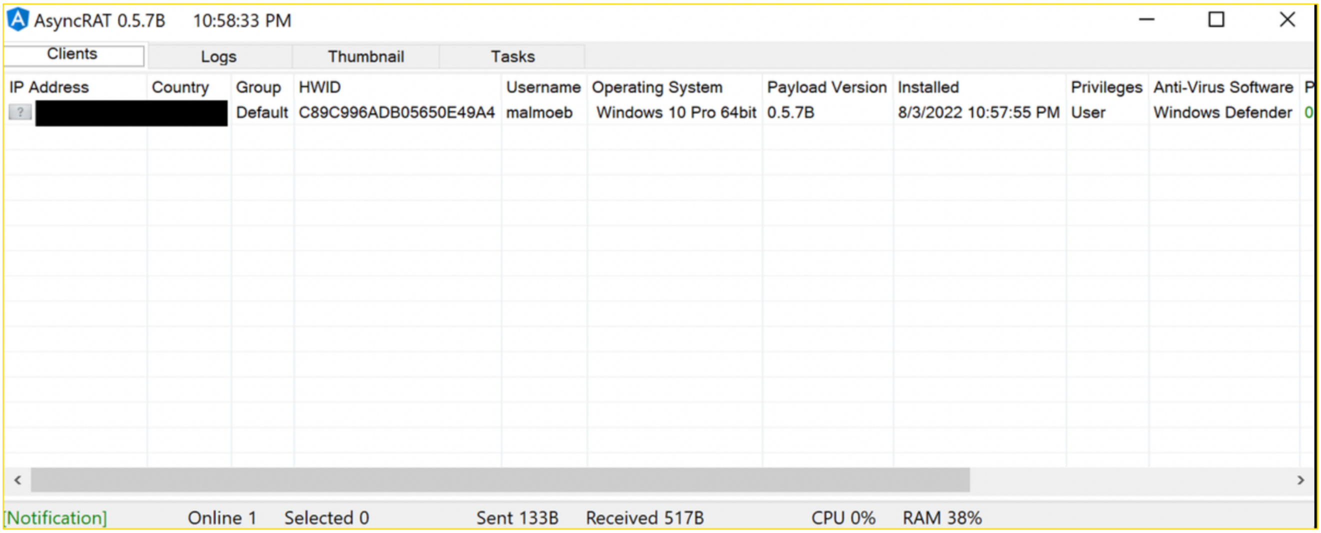
Security vendors' analysis on 2022-08-03T20:54:14 UTC

Acronis (Static ML)	Suspicious	Ad-Aware	IL:Trojan.MSILZilla.16844
AhnLab-V3	Malware/Win32.RL_Generic.C3558490	ALYac	IL:Trojan.MSILZilla.16844
Antiy-AVL	Trojan/Generic.ASMalwS.6DA3	Arcabit	IL:Trojan.MSILZilla.D41CC
Avast	Win32:DropperX-gen [Drp]	AVG	Win32:DropperX-gen [Drp]
Avira (no cloud)	HEUR/AGEN.1202836	BitDefender	IL:Trojan.MSILZilla.16844

Figure 7: VirusTotal results

Client information

After executing the AsyncRAT sample we built with the settings outlined above (in the various reports and tweets referenced so far, AsyncRAT is run at the end of a longer infection chain) on our lab machine, we see that our infected machine has reported back to the controller.



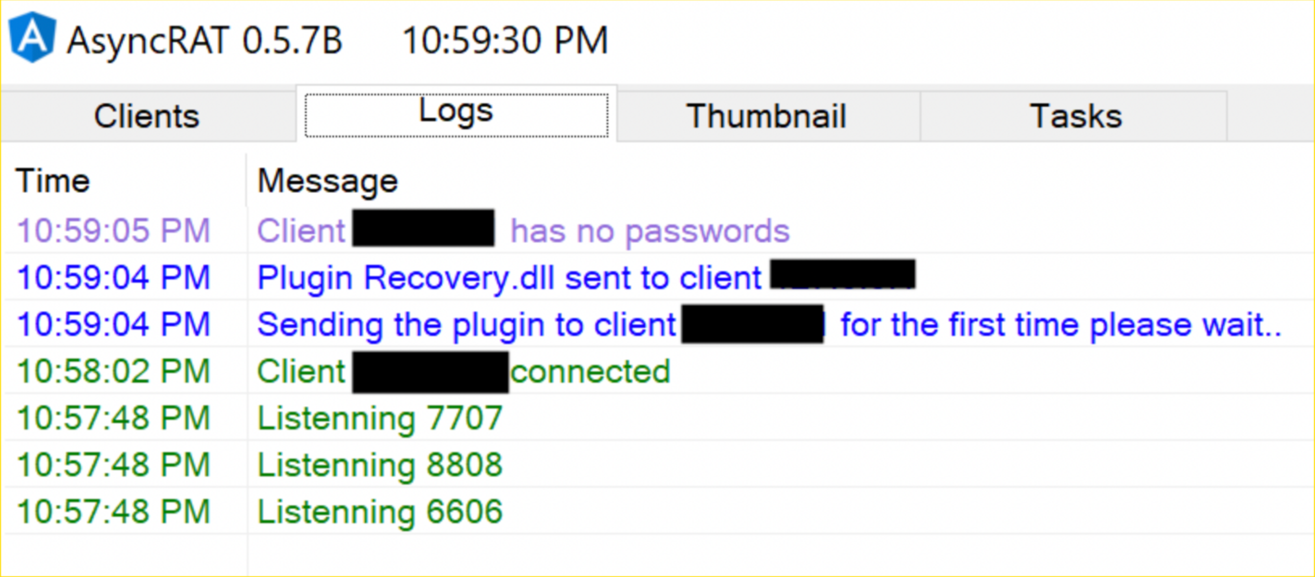
IP Address	Country	Group	HWID	Username	Operating System	Payload Version	Installed	Privileges	Anti-Virus Software	P
[redacted]	[redacted]	Default	C89C996ADB05650E49A4	malmoeb	Windows 10 Pro 64bit	0.5.7B	8/3/2022 10:57:55 PM	User	Windows Defender	0

[Notification] Online 1 Selected 0 Sent 133B Received 517B CPU 0% RAM 38%

Figure 8: Infected machines

Plugins

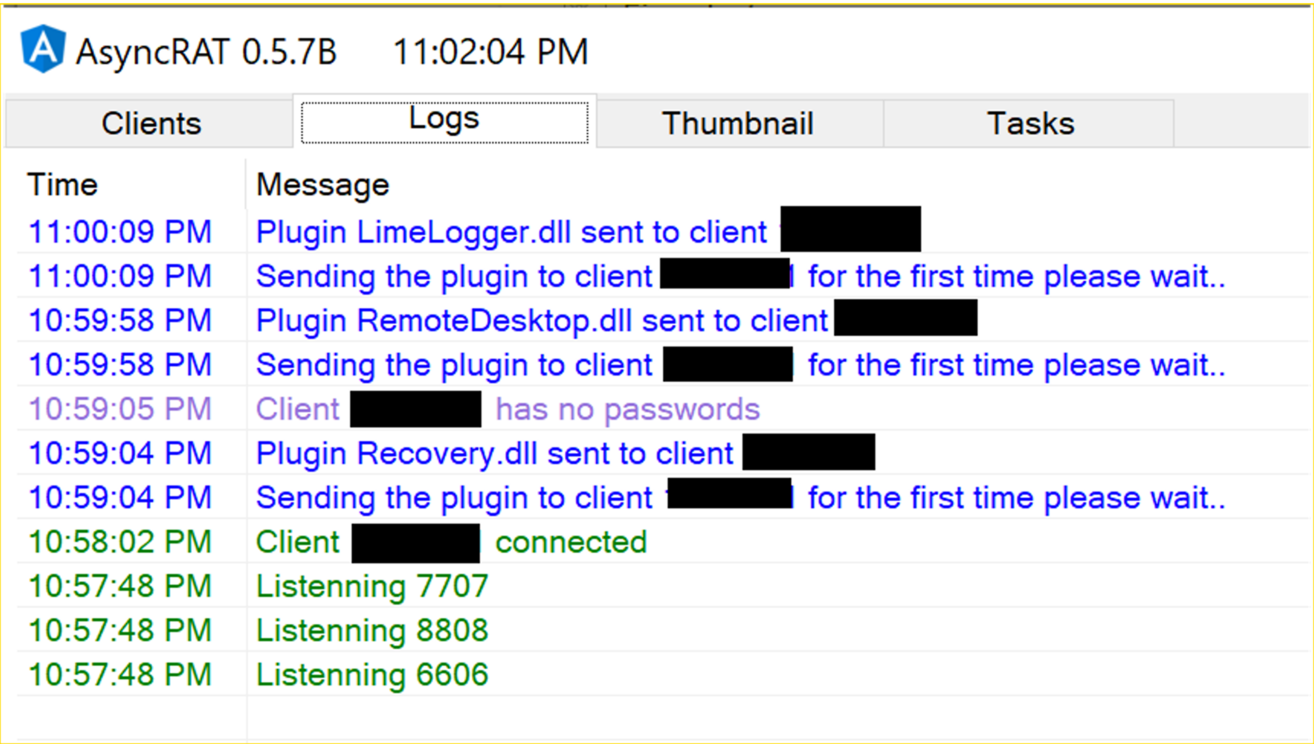
AsyncRAT now has various built-in functions that can further extend the initial access, such as “Password Recovery”. If the function “Password Recovery” is selected, we see in the logs that a DLL (Recovery.dll) was sent to the client.



```
Time          Message
10:59:05 PM   Client [redacted] has no passwords
10:59:04 PM   Plugin Recovery.dll sent to client [redacted]
10:59:04 PM   Sending the plugin to client [redacted] for the first time please wait..
10:58:02 PM   Client [redacted]connected
10:57:48 PM   Listenning 7707
10:57:48 PM   Listenning 8808
10:57:48 PM   Listenning 6606
```

Figure 9: Plugin recovery.dll sent to the client

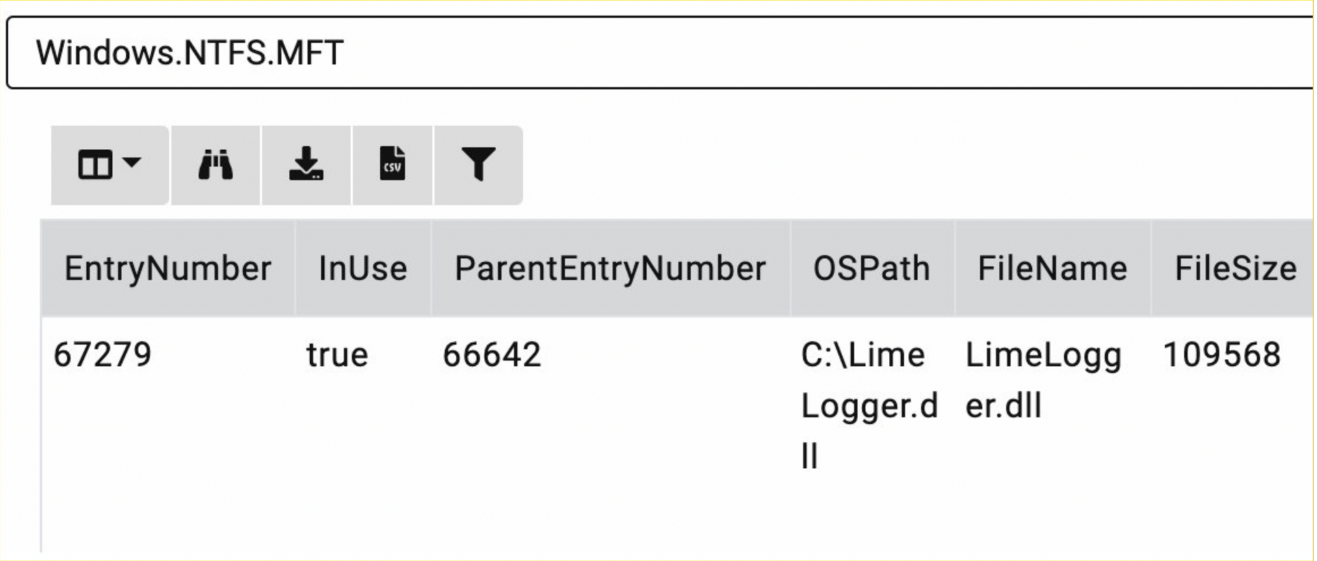
Also, using other plugins, AsyncRAT sends specific files (dlls) to the client. I have yet to look into how these plugins work under the hood, but that might be a topic for another blog post.



```
Time          Message
11:00:09 PM   Plugin LimeLogger.dll sent to client [redacted]
11:00:09 PM   Sending the plugin to client [redacted] for the first time please wait..
10:59:58 PM   Plugin RemoteDesktop.dll sent to client [redacted]
10:59:58 PM   Sending the plugin to client [redacted] for the first time please wait..
10:59:05 PM   Client [redacted] has no passwords
10:59:04 PM   Plugin Recovery.dll sent to client [redacted]
10:59:04 PM   Sending the plugin to client [redacted] for the first time please wait..
10:58:02 PM   Client [redacted] connected
10:57:48 PM   Listenning 7707
10:57:48 PM   Listenning 8808
10:57:48 PM   Listenning 6606
```

Figure 10: Sending other plugins to the client

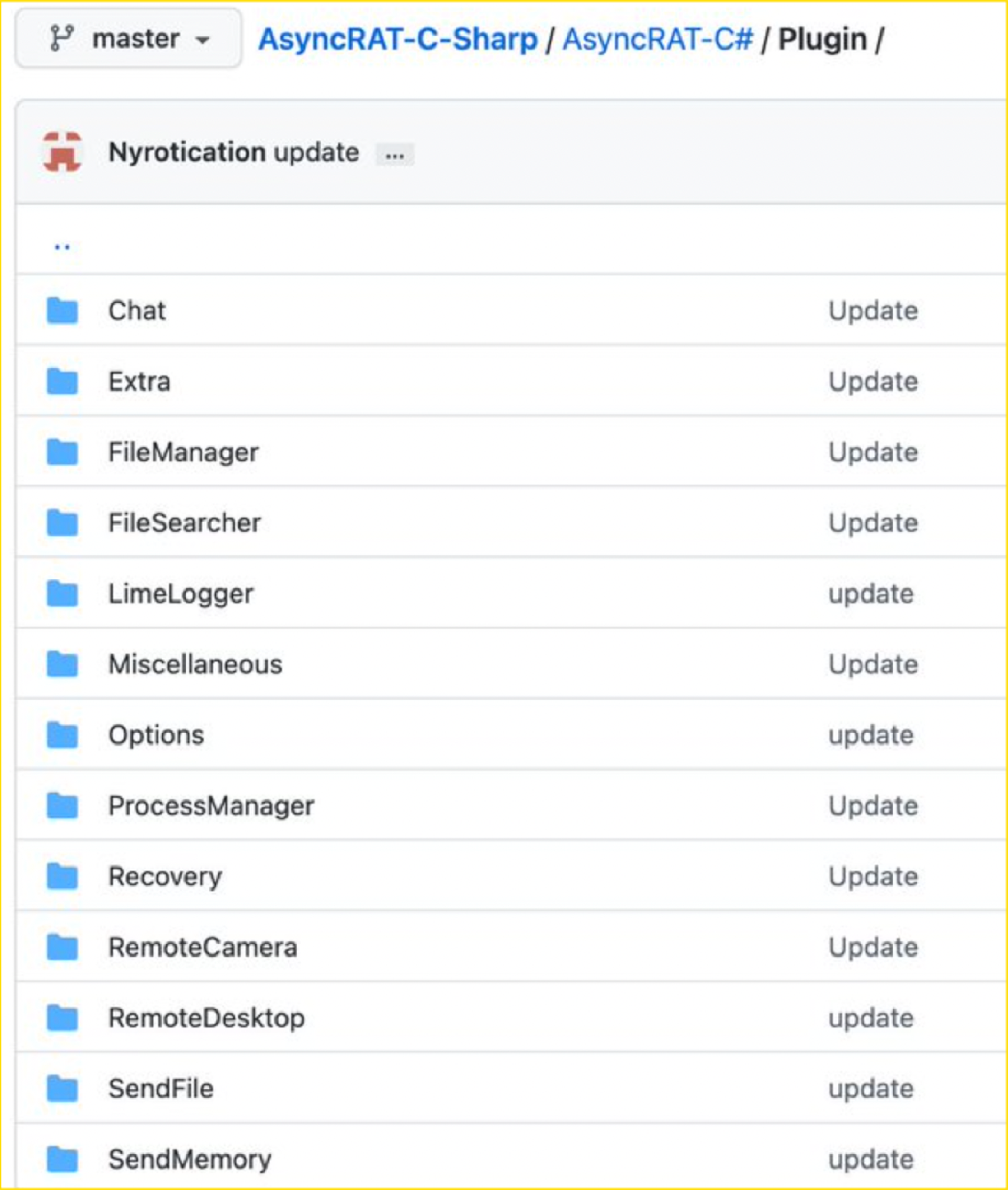
Time for Velociraptor 😍 Utilising the MTF-Hunt, we searched for the names of the plugins transfered from the server to the infected machine and found the plugins placed directly under the path C:\.



EntryNumber	InUse	ParentEntryNumber	OSPath	FileName	FileSize
67279	true	66642	C:\LimeLogger.dll	LimeLogger.dll	109568

Figure 11: Velociraptors MFT Hunt

To get a list of all plugins available within AsyncRAT, browse to the public GitHub repo.



master ▾ AsyncRAT-C-Sharp / AsyncRAT-C# / Plugin /

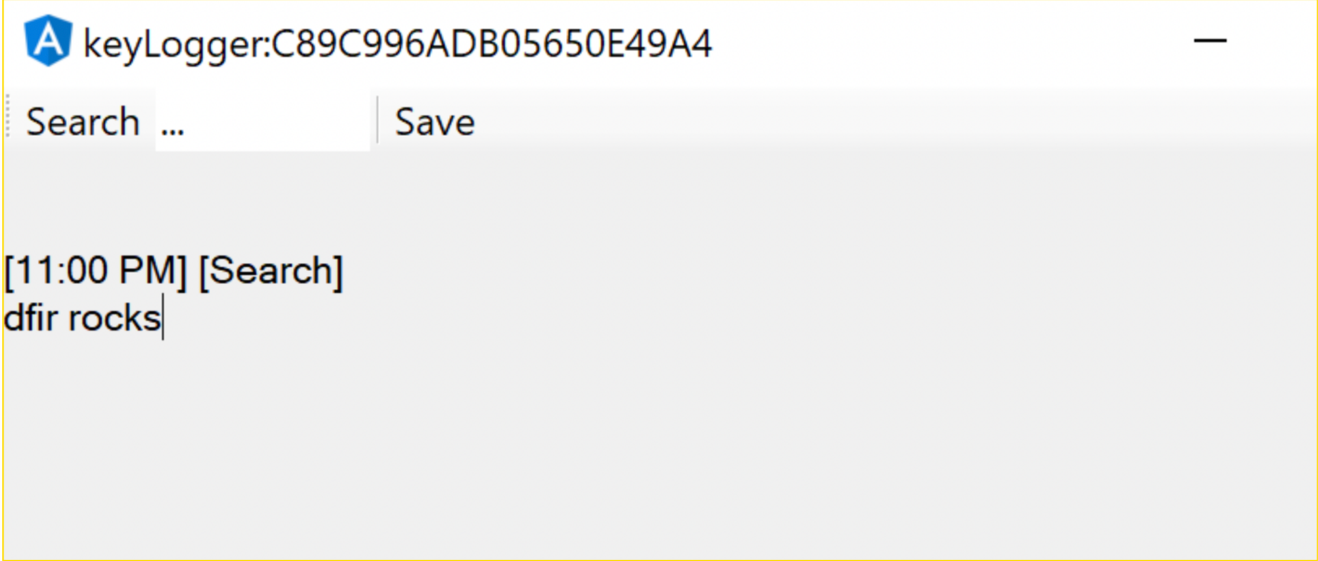
Nyrotication update …

..

Chat	Update
Extra	Update
FileManager	Update
FileSearcher	Update
LimeLogger	update
Miscellaneous	Update
Options	update
ProcessManager	Update
Recovery	Update
RemoteCamera	Update
RemoteDesktop	update
SendFile	update
SendMemory	update

Figure 12: AsyncRAT's plugins

The LimeLogger plugin for recording keyboard strokes works fine, recording all keystrokes on our test machine.

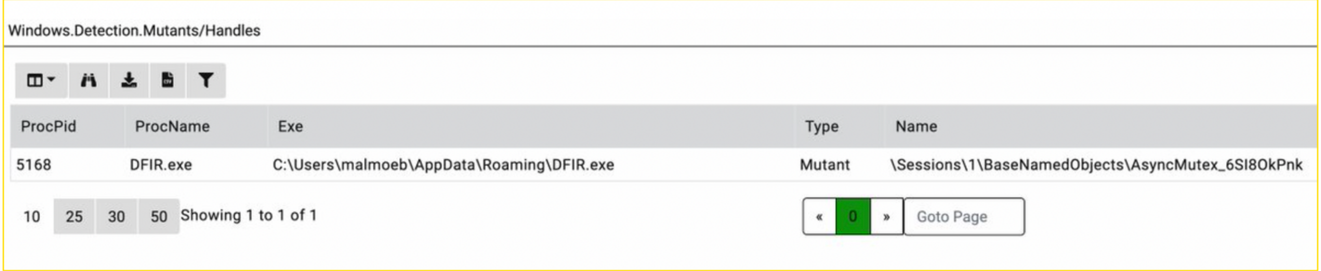


```
[11:00 PM] [Search]
dfir rocks
```

Figure 13: Logfile of the keylogger

Mutex Hunting

Using Velociraptor's Mutants-Hunt and the string "AsyncMutex_" (the prefix of the Mutex), we can also find infected machines.



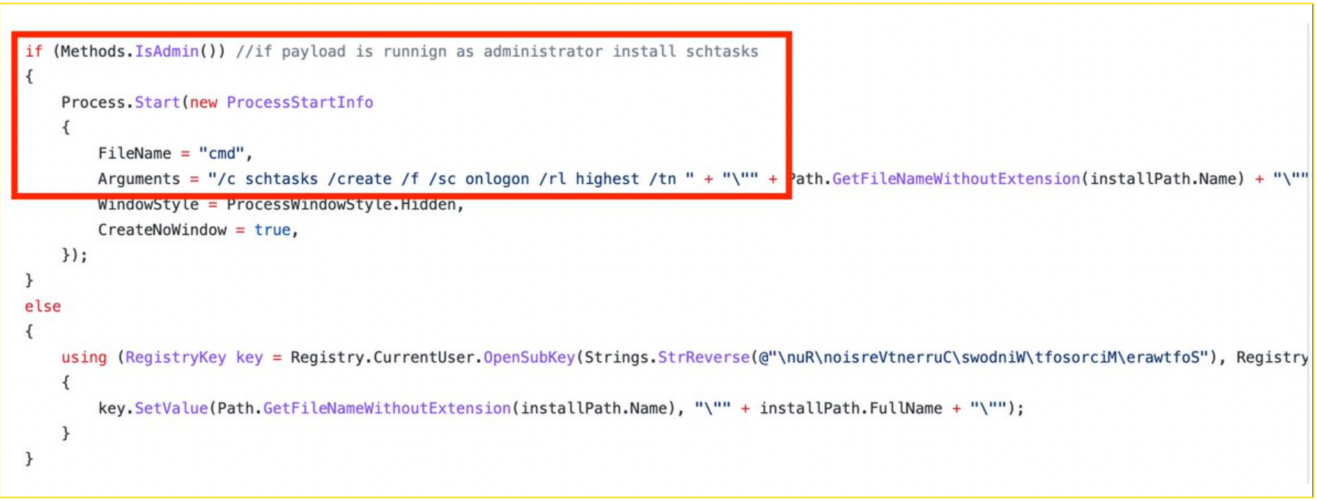
ProcPid	ProcName	Exe	Type	Name
5168	DFIR.exe	C:\Users\malmoeb\AppData\Roaming\DFIR.exe	Mutant	\Sessions\1\BaseNamedObjects\AsyncMutex_6SI8OkPnk

Showing 1 to 1 of 1

Figure 14: Velociraptors Mutants Hunt

Persistence techniques

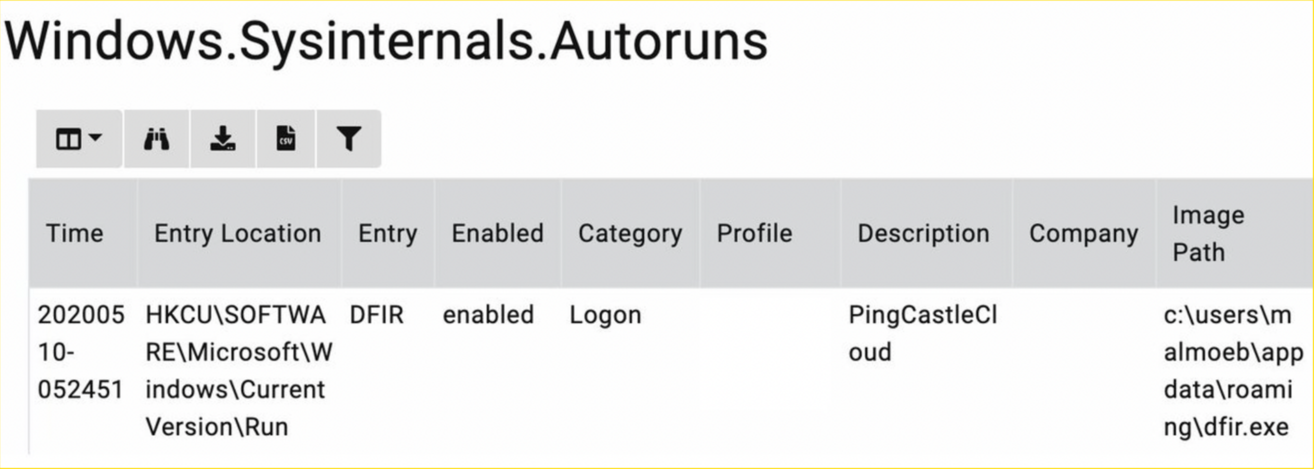
AsyncRAT distinguishes between two types of persistence. Either the installer is run with administrative privileges, in which case a Scheduled Task is created. Or a new entry in the Run Key is created when the installer run as an unprivileged user. Below is the relevant code (highlighted with the red square is the code that creates the Scheduled Task).



```
if (Methods.IsAdmin()) //if payload is runnign as administrator install schtasks
{
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd",
        Arguments = "/c schtasks /create /f /sc onlogon /rl highest /tn " + "\"" + Path.GetFileNameWithoutExtension(installPath.Name) + "\""
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true,
    });
}
else
{
    using (RegistryKey key = Registry.CurrentUser.OpenSubKey(Strings.StrReverse(@"\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS"), Registry
    {
        key.SetValue(Path.GetFileNameWithoutExtension(installPath.Name), "\"" + installPath.FullName + "\"");
    }
}
```

Figure 15: The code that creates the Scheduled Task

In Figure 16, the sample was installed as an unprivileged user, resulting in a new run key.



Windows.Sysinternals.Autoruns

Time	Entry Location	Entry	Enabled	Category	Profile	Description	Company	Image Path
20200510-052451	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	DFIR	enabled	Logon		PingCastleCloud		c:\users\malmoeb\appdata\roaming\dfir.exe

Figure 16: Run-Key

The THOR APT scanner from from Nextron findes AsyncRAT samples based on the Rule *Reversed_String* - see Figure 17 for the relevant code. Probably noisier than stealthier 🤔

```
else
{
```

Figure 17: Reversed Registry Key Path

Figure 18: Rule Reversed_String from THOR Scanner

Below is the newly created Scheduled Task (installation with admin rights) - the executable is again placed in AppData\Roaming (one of the two options in the builder).

Figure 19: The newly created Scheduled Task

QuasarRAT

QuasarRAT is another RAT we see periodically in our IR cases. It was also used against NATO facilities in March 2023, as reported by Proofpoint.

We can hunt for

- Detect the usage of standard C2 ports
- Hunting for the persistence mechanisms
- Mutexes
- Last but not least, hunting for User-Agents

Qualys has published an excellent paper, Stealthy Quasar Evolving to Lead the RAT Race, where the builder and much more details about Quasar are described in detail.

Standard C2 ports

In the client builder (which creates an executable used for the infection), the default port is pre-configured to 4782.

Figure 20: Default port

On ThreatFox, we see that a quarter of the samples kept this default port. The other samples use a different high port.

Figure 21: Default ports in the wild (Threatfox)

Mutex Hunting

We can create a “random” Mutex per client build, though random is not quite right, as the structure of the mutex remains the same. 🤔

Figure 22: Default Mutex

With the following regex, we can find these generated mutexes and thus efficiently find infected clients on the network:

```
[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$
```

Using the example of a recent sample on Bazaar, we look at the Mutexes logged by Joe Security. So besides the default port also, the default mutex values are used in the wild.

Figure 23: Default Mutex in the wild

Utilising Velociraptor and the Mutants-Hunt with the following regex:

```
(?-i)\\[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$
```

Which finds the infected machine in the lab 😎

Figure 24: Velociraptor's Mutants Hunt

User Agent

The hard-coded user agent string could be used for hunting or monitoring purposes. This User Agent ist also part of the the Sigma rule *Malware User Agent*, mainted by the Nextron Systems stuff.

Figure 25: Hardcoded User-Agent

Persistence techniques

Quasar uses the exact same persistence mechanisms as AsyncRAT, which we analyzed above in depth. A run-key entry is created when the installation is performed under an unprivileged user, or a new scheduled task is created with administrative credentials.

Figure 27: Quasar Persistence