Playing with AsyncRAT

eln0ty.github.io/malware analysis/asyncRAT/

February 16, 2022

6 minute read

AsyncRAT is a Remote Access Tool (RAT) designed to remotely monitor and control other computers through a secure encrypted connection. It is an open source remote administration tool, however, it could also be used maliciously because it provides functionality such as keylogger, remote desktop control, and many other functions that may cause harm to the victim's computer. In addition, AsyncRAT can be delivered via various methods such as spear-phishing, malvertising, exploit kit and other techniques.

We will discuss .NET code with **dnSpy** to learn how it works.

Sample overview

sha256: `8021f8aa674ce3a2ccb2e8f917ebaf5b638607447f0df0e405e837dd2e7a7ccd`

This sample is packed and I unpacked it automatically with unpac.me (online unpacker) and got this.

This is one sand box process flow

Initialization

First, Malware sleeps for 3 seconds. I don't know why but it's okay.

Second, tries to initialize all settings depending on hardcoded configurations.

(/assets\images\malware-analysis\asyncRAT\init.jpg)

Settings Details

Malware decrypts all configurations from `AES256` encryption algorithm here.

Then verifies the integrity of these configurations and returns result. If false, exits from process. You can extract values with using debugger.

Then malware checks if any of these configurations changed using `Serversignature` and `ServerCertificate` with `VerifyHash` function and returns the result. It's something like a water mark in coding :)

Config Decryption

I'm not an encryption nerd but I will try to explain as I can and we don't need to understand how it works to continue our analysis but I would love to give some help to learn some useful things. If you don't care just scroll the whole topic and go to **Mutex creation**. Let's start with `Key = ejFjc0p0QWtudENHVTdsakhjTExYbm1KM1RqbTVUMlA=`.

It converts `key` from Base64 then encoding to **UTF8** so now `Key = z1csJtAkntCGU7ljHcLLXnmJ3Tjm5T2P`.

Deriving keys

This link gives you a complete definition about this encryption algorithm. The usage of it is to derive a new key in run time from our previous key.

To solve it, we have to focus with its parameters `dec_key = PBKDF2(key, Salt, iterations)`

AES256

Then use this dec_key with aes256 algorithm to decrypt all configurations.

This method divides the given config, like `ports` into 3 sections:

Data[:32] -> HMAC-SHA256 value

Data[32:48] -> IV

Data[48:] -> Encrypted bytes

Script

This python script automates all decoding components.

```
# 1) use PBKDF2 to derive the decryption key and initialization key used for sha
# 2) calculate sha256 of data[32:] and compare it to the embedded sha256 hash
(data[:32]) (We don't care here)
# 3) iv = data[32:48]
# 4) aes_dec(key, iv, data[48:])

# pip install backports.pbkdf2
# pip install malduck

from backports.pbkdf2 import pbkdf2_hmac
from base64 import b64decode
from malduck import aes, unpad

salt =
b"\xbf\xeb\x1e\x56\xfb\xcd\x97\x3b\xb2\x19\x02\x24\x30\xa5\x78\x43\x00\x3d\x56\x44\xd2

key = b"ejFjc0p0QWtudENHVTdsakhjTExYbm1KM1RqbTVUMlA="

config = {
    "Ports":
"UGCInR8TOWCBkQI6fVXrRZ4Yj+b4OvMqcvbx3n2pTLIpcwWtvmX+PX6uN7uIsx65cuUHbVopkDdPuRbLHd6jf

    "Hosts":
"k/33hCqQ1vnvaz3j8VvjdZRXF/poiYruJfX1WbFuFhwXYuNriBFrqyi0fQfk4xN0LS85PC6oOtCuLYarjJSnL

    "Version":
"WG0EkFzynw3wCeMtt128RLUZgT6BSNw7pqLDg9XUMRmpx5WpQw1ZN64GLHYrP/h47iM2KImVVeY0wAT1RqMVV

    "Install":
"3/TL2kdA5ptdHUR1gfeiPmkurKrJsw3BjJ7njALFi+ouT64Tx5oE1P7U7NktNpWfBZVmmjxeR/xSyR14NdEPc

    "MTX":
"7vyshlirEg6SwhKPRttI85LoRXYLoFWLzaDM4h57MqKcy9iihijskYVbiDhhZu5qzqRxMBX5DpJ6dAfancdQ8

    "Anti":
"fvHzWJyCKwkBHk/dOoyPPC5w+F3GyNg0t7NAj8VXjA2b0ntbSqH11xvQACf2jGX7VSLAd6BjykqqQIJAb98Ve

    "Pastebin":
"B52OeJUAfsMHW3Ea2wBUni41OckwUyCtHz3yHsDSn9XjE4U+ncvS0Kmik61ZnDWTm+oNBPoQaDb5PHqfInPGX

    "BDOS":
"++zHWqz0o5rkma5tjGrmNMSXzvLTZVOFmlOz4lhTPTPejjFLjqH/rhhciAYgm+Mq5bOazkPYeFGYC8q5I47wV

    "Group":
"fwbqIWwfsG6vrljdbLznhYHm5g+qylXiJVparVYZ5s61hXK84/sQMNn6fTH09rZ+MeWdbYV1AhcKtEpQzJ6I5

}

key = b64decode(key)
dec_key = pbkdf2_hmac("sha1", key, salt, 50000, 32)

for k, v in config.items():
    data = b64decode(v)
    iv = data[32:48]
    decrypted = unpad(aes.cbc.decrypt(dec_key, iv, data[48:]))
    print("{}: {}".format(k, decrypted.decode("utf-8")))
```

After running the script, we have a clean config.

```
key             <-      "z1csJtAkntCGU7ljHcLLXnmJ3Tjm5T2P"
ports           <-      "6606,7707,8808"
Host            <-      "jeazerlog.duckdns.org"
version         <-      "0.5.7B"
Install         <-      "false"
MTX             <-      "AsyncMutex_6SI8OkPnk"
Pastebin        <-      "null"
Anti            <-      "false"
BDOS            <-      "fasle"
Group           <-      "gta"
```

I want to note that the malware is also extracted **Hwid** while execution, and I got its value using the debugger `Hwid = 1021C7B642607CE65116`

Mutex

The bad boy tries to make Mutex handle with MTX value which extracted from Settings to prevent the duplication of the process `MTX = "AsyncMutex_6SI8OkPnk"` and tells windows "end the duplicated process".

Anti Analysis

We are lucky because malware doesn't use any anti-analysis technique according to `Anti = fasle` in Settings class.

but I will explain what if a malware developer chooses a difficult path with analysis `Anti = true`. The malware developer would have used five methods to make it difficult for the malware analyst to use.

1. VM detection: malware searching in **Manufacture Model** for keywords like `VIRTUAL` or `vmware` or `VirtualBox`.

2. Debugger detection: Check if the debugger is present to stop the process.

3. SandBox detection: Tring to get a handle from **SbieDll.dll** that belongs to every sandbox.

4. Small Disk detection: Most secure labs for malware analyzers such as virtual machines contain a small disk.

5. XP windows detection: You know, Nobody uses XP today except for malware analysis or something.

Let’s move on to the next step in our main function.

Install

Once again, we are in luck, the malware author decided not to use any persistence mechanism according to `Install = fasle`.

But I will explain the hard path again, What if `Install = true` in Settings? Let’s go…

The first thing is that the malware checks the path it is running on, and if it is not the same as the path in the settings, the running process is erased.

The malware creates a `.bat` file in the `%temp%` to run a new process, created in the hard coded path `%AppData%`, then deletes itself.

Persistence

The malware checks if a process has administrator privilege to perform a schedule task every time a user logs on to run or has a normal user privilege to modify the `Software\Microsoft\Windows\CurrentVersion\Run` subkey to be added in the list of startup processes.

BSOD

Malware passes this step in the main function because `BDOS = false`.

otherwise it would have verified that the user is an administrator and the operating system has been switched to the critical state.

To learn more about `RtlSetProcessIsCritical` and what its risks are, this link explains in-depth.

Finishing configurations

So far, the configuration has been done and the malware will run almost forever.

The next step will stablish the connection with C2 server.

Connection with C2

I won’t explain the code too much at this level of analysis because it’s a development problem, I’m just explaining what’s going on.

The malware creates an infinite loop to connect to C2 and the first thing it does is check if it’s already connected or not, then sleeps for 5 seconds to free up resources so windows won’t crash.

I’ll explain a little bit what happens when malware disconnect.

First, It calls a Reconnect function to dispose any packets between each other.

Then it initializes a new tcp client connection through the TLS protocol for secure connection. You can check the code by yourself. -_^

Server side operations

When the victim runs the malware in any way, whether by phishing mail or otherwise persuaded by another method, it appears to the hacker that he has run the program, and here the victim is completely controlled in a terrifying way, some of which are shown in below.

Conclusion

Malware declares all settings **AES256** then trying to connect victim machine to C2 server. From this point, all commands come from the other end of the world through the C2 server which were not embedded in the code.

Finally, I hope you had fun and learned something new. See you in another analysis report.

IOCs

Hashes

Packed: 8021f8aa674ce3a2ccb2e8f917ebaf5b638607447f0df0e405e837dd2e7a7ccd

Unpacked: bc61724d50bff04833ef13ae13445cd43a660acf9d085a9418b6f48201524329

C2s

jeazerlog.duckdns.org:6606

jeazerlog.duckdns.org:7707

jeazerlog.duckdns.org:8808

MUTEXs

AsyncMutex_6SI8OkPnk

REGs

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run