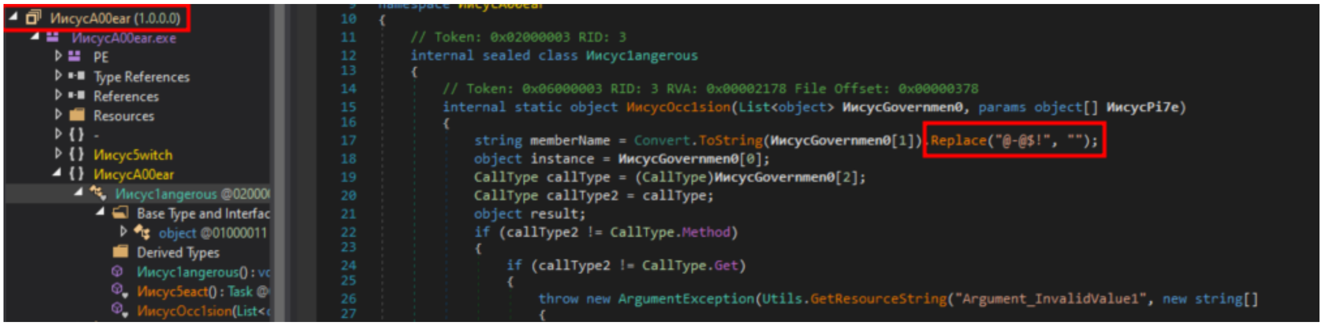
AsyncRAT: Analysing the Three Stages of Execution

medium.com/@hcksyd/asyncrat-analysing-the-three-stages-of-execution-378b343216bf

Hack Sydney

February 8, 2023



```
namespace ИисусA00ear
{
    // Token: 0x02000003 RID: 3
    internal sealed class Иисус1angerous
    {
        // Token: 0x06000003 RID: 3 RVA: 0x00002178 File Offset: 0x00000378
        internal static object ИисусOcc1sion(List<object> ИисусGovernmen0, params object[] ИисусPi7e)
        {
            string memberName = Convert.ToString(ИисусGovernmen0[1]).Replace("@-@$!", "");
            object instance = ИисусGovernmen0[0];
            CallType callType = (CallType)ИисусGovernmen0[2];
            CallType callType2 = callType;
            object result;
            if (callType2 != CallType.Method)
            {
                if (callType2 != CallType.Get)
                {
                    throw new ArgumentException(Utils.GetResourceString("Argument_InvalidValue1", new string[]
                    {
```

Hack Sydney

Feb 8

·

8 min read

Michael Elford

```
Michael is a cybersecurity professional that is passionate about malware analysis and
reverse engineering.
```

In the second half of 2022, I led an investigation that involved a suspicious registry key, comprising a value that would run a potentially malicious executable file.

Further analysis of the executable file resulted in confirmation of the file being malicious.

In this blog post, I aim to share the findings from the investigation, with the goal of making the audience aware of the techniques that can be used to effectively analyse malware in general and specifically analyse ASyncRAT.

ASyncRAT Analysis

```
Note: All malware analysis should be carried out in a sandboxed environment.
```

Tools used

This is the list of tools that were used for this analysis:

Stage One

Starting the analysis of Stage One of the malware, the first step was to check the executable in PEStudio, which confirmed that we were dealing with a .NET file. Furthermore, we interpreted the fairly high entropy as indicative of malware packing, where malware adds complexity to stay undetected by security solutions.

PEstudio file overview
Our next step was to peruse the strings of the malware, and take note of anything strange we may want to look into a little bit deeper later on.

PEstudio file overview
Examining the strings, we immediately identified some strings that appear to be **obfuscated**. As a note to ourselves, we highlighted that we may need to deploy a 'find and replace' later on.

Going Further

After determining that the file was written in .NET and making note of the obfuscated strings, we then turned to dnSpy, a .NET debugger.

One of the first things we noticed was a naming discrepancy. The name of the executable on the host and the name of the executable shown in dnSpy are completely different. The reason as to why the executable having a completely different name is of interest is due to many threat actors that use malware rename them so that they do not draw as much attention and can be hidden on the host and quietly run in the background so that even if the user does notice the something strange is going on with their system, they will skip over the file thinking that it is normal and doesn't raise any alarms with the user and their AV.

Looking through the code, there is a replace function for a string we had noted earlier. Now that we are certain about what is replacing the obfuscation pattern, we can check what those strings are, which could possibly give us a clue as to what is happening.

Executable name and replace function

Pulling on the Thread

For analysing the obfuscated strings, we used **CyberChef**. In CyberChef, we used the `Find/Replace` feature to perform the replace function found in the code. This enabled us to see the clear text of *what* was obfuscated which resulted in getting a clearer understanding of what was going on *within* the code.

The end-result of the above exercise gave us the de-obfuscated version of strings which contained “Load” and “Invoke”. This showed that something was going to be loaded and executed (“invoked”) by the code contained within the executable.

If at any time you are unsure what actions these methods perform, the best thing to do is Google around and find some docs. It may be time consuming at first but it is an important part of analysing malware.

De-obfuscated strings
At this stage of the analysis, we had accumulated enough findings to start sketching out the trajectory of the malware’s execution flow. We had identified the malware was loading and executing *something,* and that the malware was packed and contained executable data that we would see in the next stage of analysis.

Hunting for the Hidden

The tool **Garbage Man** is used for heap analysis — a technique that allows analysts to intervene and observe as the .NET malware is unraveling itself.

Analysing the malware in Garbage Man aided in extraction of hidden data that composed the *second* executable within the *first* executable.

In order to further analyse the malware, we executed the original malware found on the host in Garbage Man. To get started, we added in a delay of 300ms with 10 snapshots taken at 100ms intervals.

We added in the delay to allow time for the process to start running and added the extra snapshots and intervals so that we can compare the difference between the intervals if we need to.

Once Garbage Man had executed the file, we had these options:

- Sift through the heap manually, and look at the strings and values that are contained within the executable
- Use the search function to look through all the snapshots. Specifically looking for bytes that start with the value of — which is the for executables and dll files, and then order them by size in descending order. The search function gives us a list of results, and we can save the binaries with MZ headers and save them for further analysis.

Using Garbage Man to search for executables and save the binary for analysis
When saving the executable, we chose an executable with the large size value and due to us looking through all the snapshots we were getting repeats of the same executable, we could then save it as “stage2.exe”. We then pivoted back to PEstudio to check the file. On examination, our second hidden executable yet again turned out to be a .NET file.

PEstudio of stage2.exe

Stage Two

Delving into our stage2.exe, we found that checking its strings highlighted some noteworthy Win32 APIs that malware leverages. Some APIs that jumped out were connected with manipulating processes [T1055], specifically in this case, process hollowing [T1055.012] — an adversarial technique to create a paused, legitimate process then swap out its memory with malicious contents.

Some of the APIs and Strings that could indicate Process Hollowing
As our stage two was in the .NET world again, we used dnSpy to dig a little deeper. When opening up the executable, we were met with a *lot* of information we had to get through. Like the first stage, the second stage's naming convention was also suspicious.

Stage 2 dnSpy
Examining dnSpy's output, we discovered a section completely different to the other parts of the code due to it being obfuscated, and seemingly performing a function in memory — both of which are techniques used to evade detection [T1140].

Obfuscated section in Stage 2
We used Garbage Man once again to sneakily observe how the malware unraveled itself. However, if you want an alternate technique to de-obfuscating this part of the code, you can use de4dot which may also lay it out in an easier to read format.

Next, in Garbage Man the objective was to retrieve the hidden stage. Our prior technique could be deployed here again, executing the binary and searching for the MZ headers.

We found an executable and saved it as "stage3.exe". We then packaged this up and shipped it to **PEstudio** (There is also information for procexp.sys contained in this but we are not going to focus on that at this time).

Searching for MZ header in stage2.exe

Stage Three

When checking the third stage executable in PEstudio we observed it to be a .NET file and that it contained strings and APIs that suggested it was trying to make a network connection to an external source — like the API . This was a big hint that this was a Remote Access Trojan; I smell a RAT [TA0011]!

APIs showing stage3 trying to connect to external sources

After we had a quick look at stage three's strings through PEstudio, we hopped back over to ol' reliable dnSpy. Trawling through stage three, we identified that AES encryption was used to encode the port and IP/domain that made up the adversary's command-and-control (C2). We were for sure dealing with a RAT, and we wanted to gather the IoCs associated with this encryption.

AES encryption

We set a breakpoint in dnSpy where the AES decryption was performed, and then run the debugger for this to find the decrypted string. Using a 'stop-start' technique with breakpoints in a debugger allows a malware analyst to halt the malware at opportune moments as it decodes or decrypts itself. This allows us to understand what the malware is doing as it is doing it.

dnSpy breakpoint

When we got to our breakpoint it paused the malware's execution. Now, we gingerly allowed the malware to progress one step at a time by using the step over button. We did this until eventually the malware began its decryption phase, and we were first gifted the ports used by this RAT.

Ports decrypted in dnSpy

When we stepped over the different sections we were able to see that for this RAT it was using ports "6606, 7707, 8808" and, eventually we unraveled the RAT until it delivered us the adversarial server it reported back to: "109.206.241[.]84".

Defending against RATs

Our remediations will vary, but the first step is to recommend blocking this public IPv4 at the organisation firewall level so the RAT can no longer beacon out.

The advice for defending against RATs is really advice for malware overall:

- Chief advice is to train staff to check what they are downloading. Whether it's from an email attachment, or a site that promises to make their computer run faster if they run something, is the best and first layer of defense we'd advise you prioritise.
- Leverage network telemetry to try and monitor for unexpected activity. Whether this is VPN, firewall, or more security-focused netflow, keeping an eye on in- and outbound traffic will help spot anomalous behaviour. Moreover, cultivating a well-maintained deny-list for can only be a good thing.
- Good security tools are the perfect enemy of our adversaries. Deploy what works for your organisation, just please deploy . A machine without any security solutions is a machine that presents an unmanageable, unacceptable risk to your business.

Big thanks to and for help and guidance. Hopefully this writeup has helped show examples of some good tools and techniques that can be used to help decrypt and understand what is going on with other .NET executables in the future.