**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.

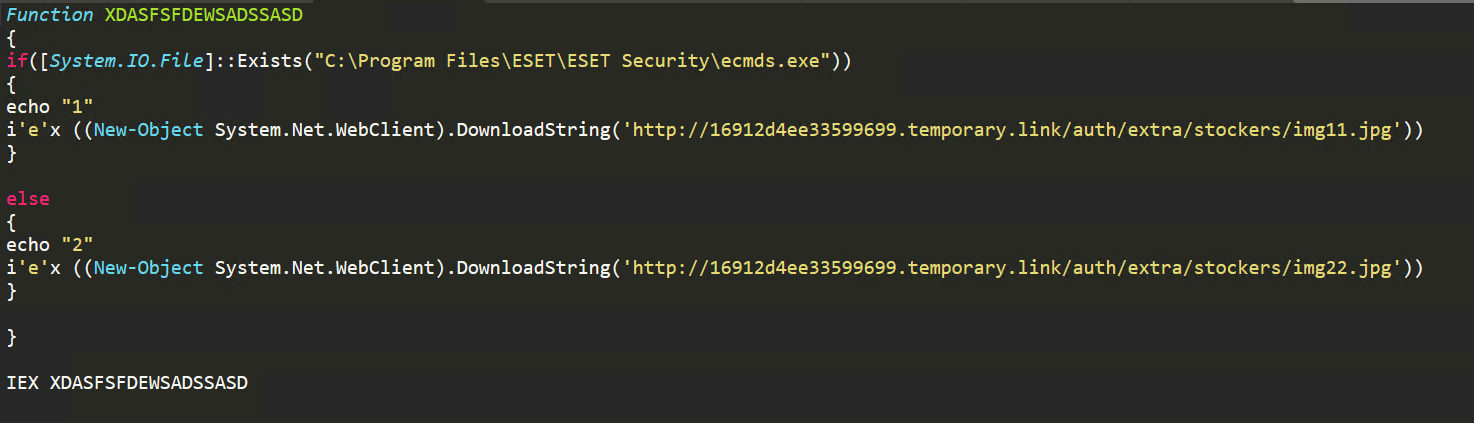


Figure 1 VVpost2 PowerShell script

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.

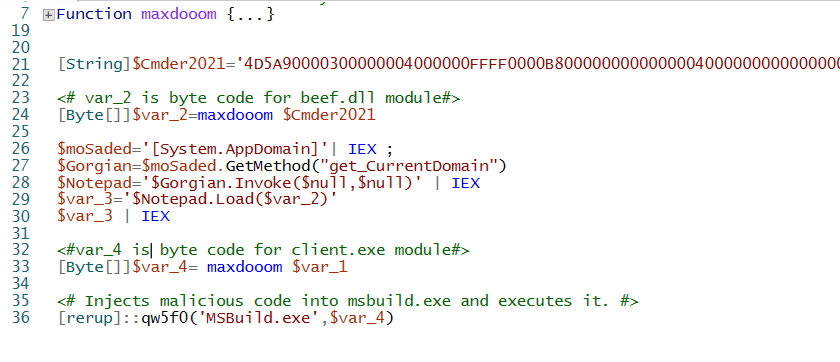


Figure The PowerShell script containing .NET assembly bytecodes

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.

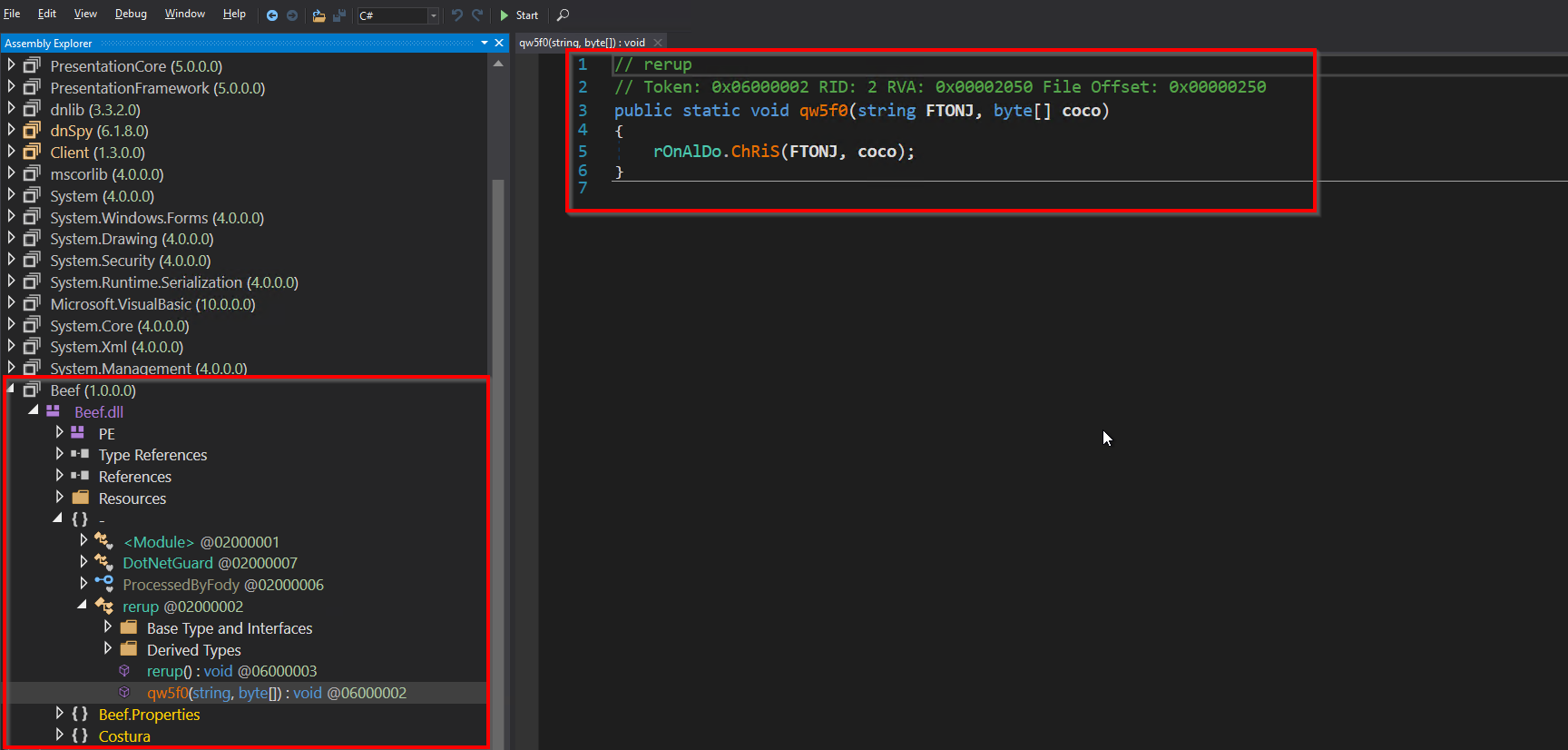


Figure Beef.DLL – Ronaldo.Chris method overloads the qw5f0 method.

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 almost identical from stage 3 onwards. This calls for further investigation to look into 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.

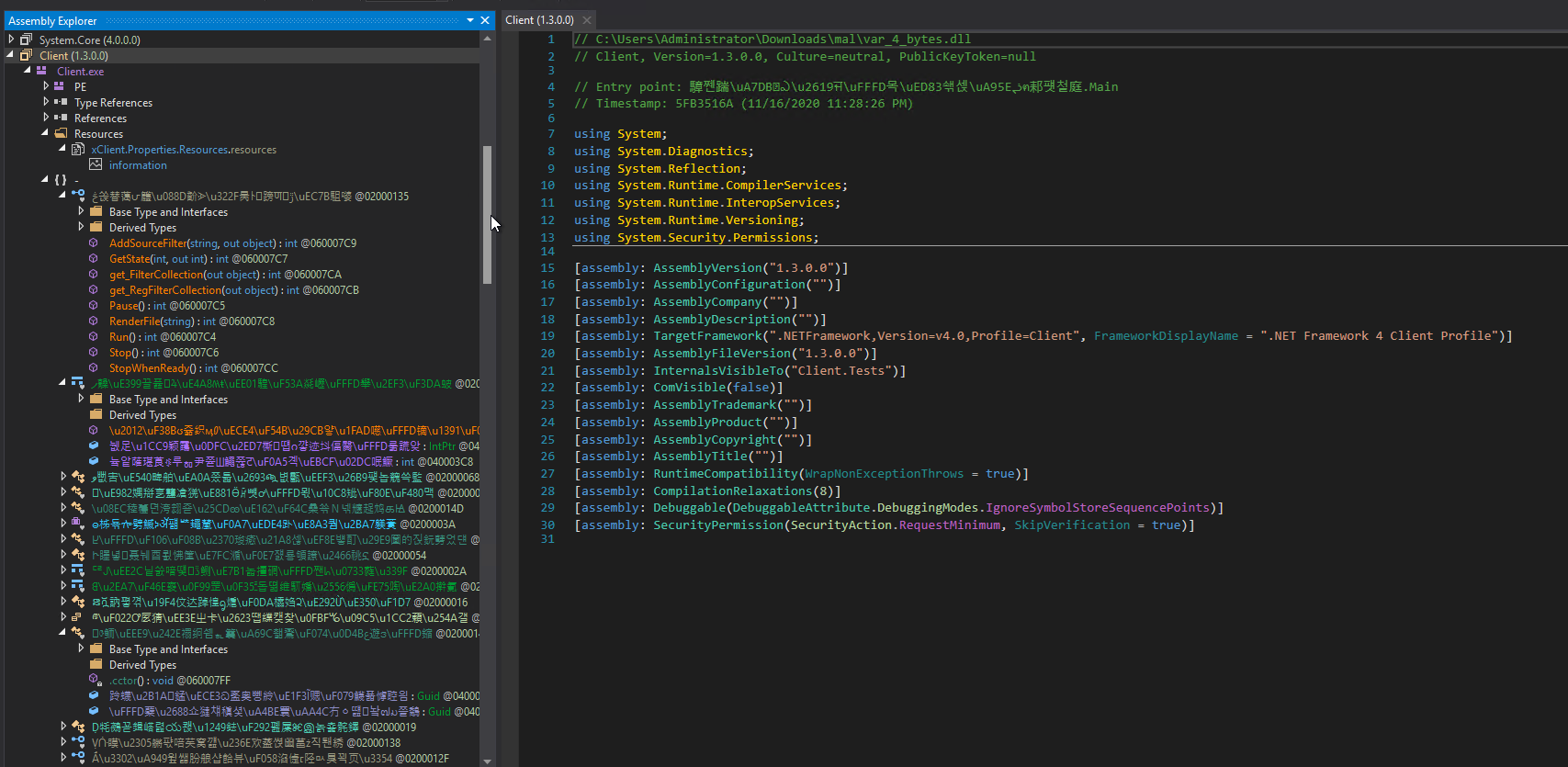


Figure The obfuscated Client.exe assembly

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

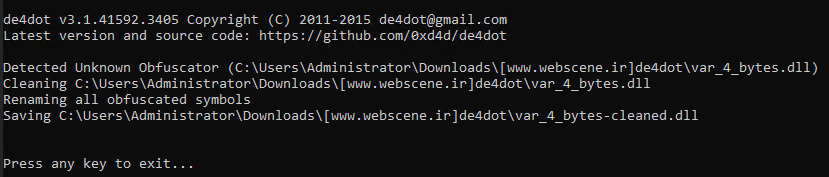


Figure de4dot Deobfuscation output

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

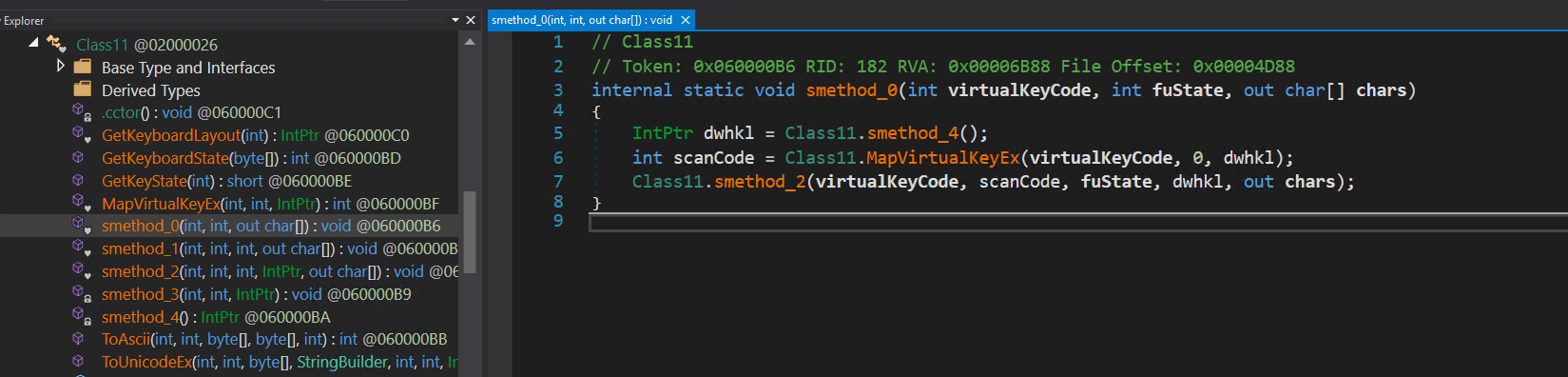


Figure Methods showing evidence for Keylogger

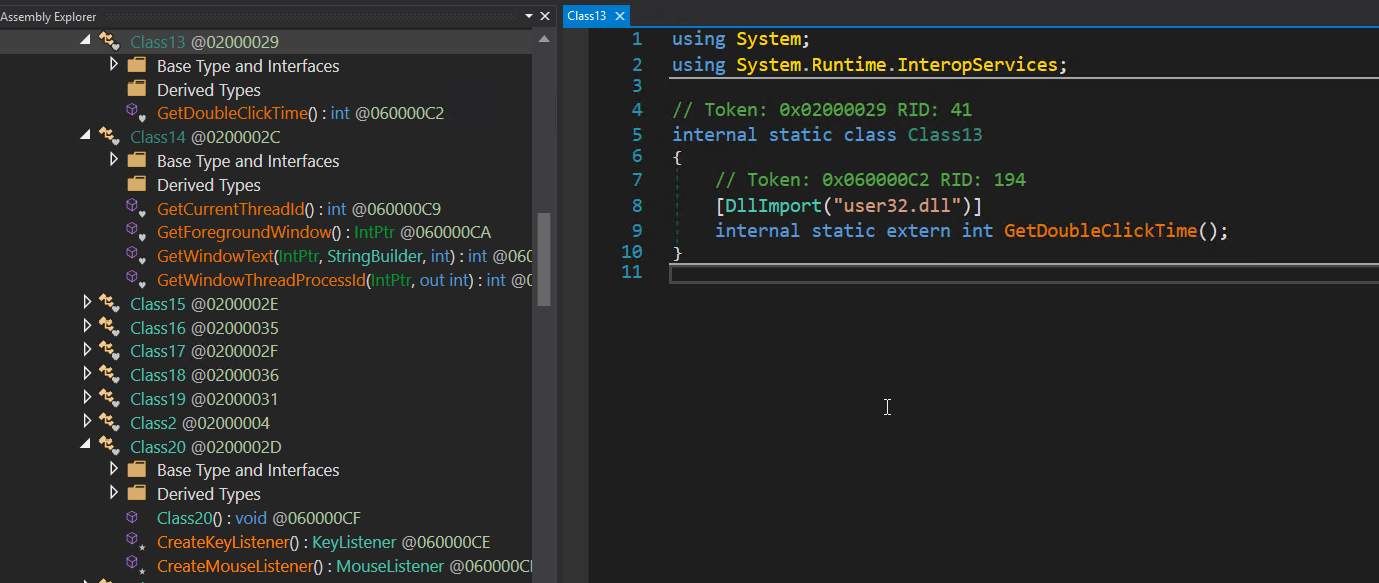


Figure Methods showing evidence for Keylogger with advanced capability

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.

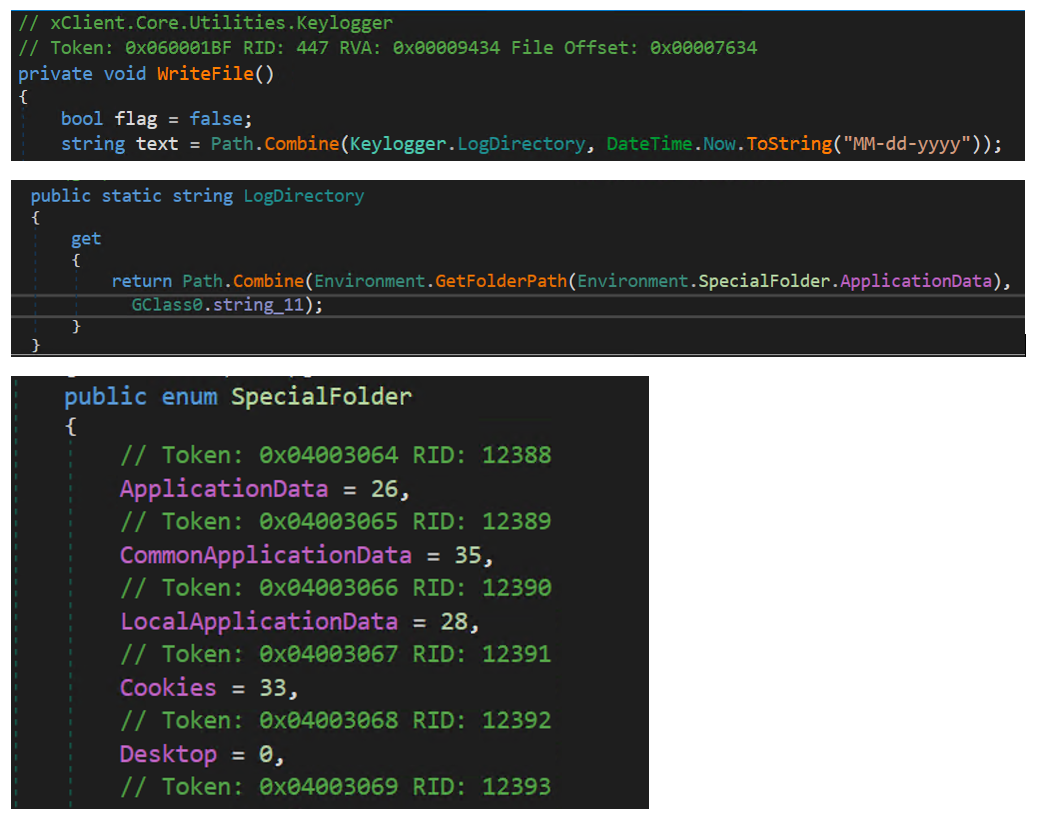


Figure Keylogger saves the information in log file

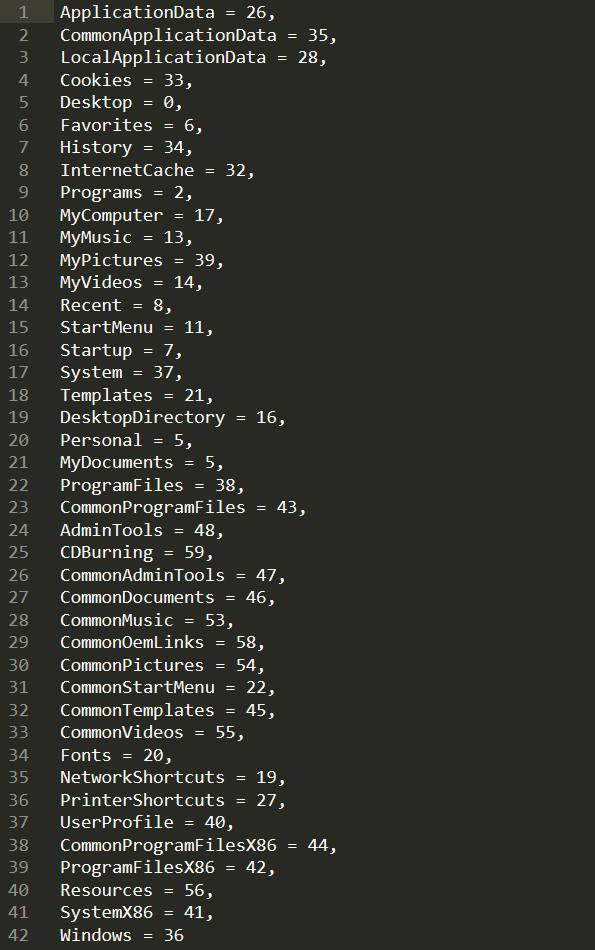


Figure List log directories used by the Keylogger

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.

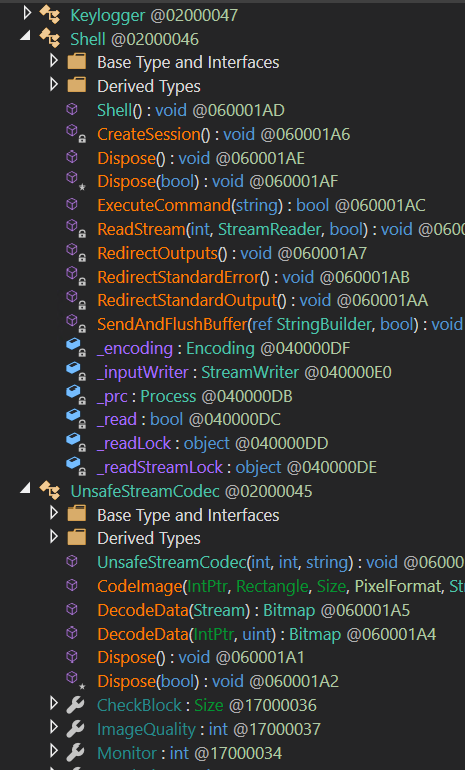


Figure Reverse shell functions in Client.exe assembly

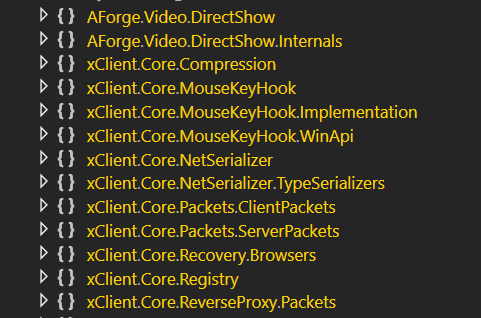


Figure Reverse proxy, remote access functions