Data Exfiltrator

blog.reversinglabs.com/blog/data-exfiltrator

Threat Research | July 15, 2021

Blog Author

Robert Simmons, Independent malware researcher and threat researcher at ReversingLabs. Read More...

Summary

Over the past year, a major change in tactics employed by ransomware adversaries is to exfiltrate data from the victim's environment. The data then serves as the material for an extortion threat on top of the ransom for encrypted data. This additional tactic became a trend followed by most major ransomware families early last year, 2020 [1].To support this tactic, some ransomware operators have added a specific type of malware to perform this exfiltration to their intrusion set [2]. The five samples analyzed here perform this type of data exfiltration. They upload a set of files from the victim's computer to command and control servers hosted on IP addresses 51.81.153[.]212, 51.161.82[.]135, and 51.77.110[.]6. All of these IP addresses are owned by OVH SAS, a French hosting company. The malware follows the exfiltration with a single line PowerShell command that stops the malware's running process and then deletes the malware file that was executed. The malware has a type of anti-analysis behavior called "Relocate API Code" according to the Malware Behavior Catalog's [3] categorization [4]. The malware reads a copy of system DLLs into memory and resolves imports from there. This causes a problem for debuggers such as x64dbg [5].

Interestingly, these files share code with an earlier malware sample with completely different capabilities. This earlier file has been observed alongside TrickBot, CobaltStrike, and ransomware [6]. This earlier malware additionally uses the same anti-analysis technique, but does not exfiltrate data. It has the capability to download a CobaltStrike beacon and execute it [7]. In addition to this overlap in code and behavior, the command and control (C2) infrastructure domains are registered via the same registrar. Also, the C2 IP addresses are owned by the same hosting company, OVH.

Anti-analysis Trick

Relocate API Code

The first behavior one observes when running these samples in a sandbox or in a debugger is that at the point where the imports are resolved an exception is raised. Debugging past this point is not possible without circumvention of an anti-analysis trick. This circumvention starts by examining the first encoded string the samples decode. Encoded strings are found in two general forms in these samples. First is with all the rest of the strings in the file in the .data section. Some of these can be seen in Figure 1 with one example highlighted.

```
.data section started  {0x140009000-0x140009b08}
140009000  data_140009000:
140009000  fb fc fe ff aa 76 72 6f-73 66 6a 4d 71 75 6f 39-7c 75 7e 00 07 6d 4f 72-78 78 78 78 00 00 00 00  .....vrosfjMquo9|u~..mOrxxxx....
140009020  data_140009020:
140009020  fb fc fe ff aa 4e 71 7d-6d 71 72 68 37 3e 38 3b-2c 35 65 78 7e 75 81 8a-87 35 64 6b 38 4a 4a 49  .....Nq}mqrh7>8;,5ex~u...5dk8JJI
140009040  4c 58 3e 76 6f 78 58 57-4d 45 67 97 98 95 8f 82-91 8f 79 98 a4 60 67 66-6b 63 69 6d 58 61 85 83  LX>voxXWMEg.......y..`gfkcimXa..
140009060  90 8a 8a 6b 60 ad ab ae-a9 65 8d ac ab b4 b9 74-6c b1 b7 c2 b3 c0 c4 b7-83 85 84 87 86 8c 8a 94  ...k`....e.....tl...............
140009080  7c a0 c6 d1 cf ce c7 92-9c 98 94 97 96 9d 9b 9b-9f 9b 9f a1 a2 91 b7 df-d9 d8 ea e9 e7 e7 a9 b4  |...............................
1400090a0  aa b0 ac b4 a0 d4 e3 e9-e5 f7 ef b6 bd bc c1 b9-bf c3 00 07 6d 4f 72 78-78 78 78 00 00 00 00 00  ...................mOrxxxx.....
1400090c0  data_1400090c0:
1400090c0  fb fc fe ff aa 48 47 57-00 07 6d 4f 72 78 78 78-78 00 00 00 00 00 00 00                          .....HGW..mOrxxxx.......
```

Figure 1: Encoded Strings

Notice the "mOrxxxx" characters that trail each of the encoded strings. These trailing characters will be examined below.

The other location where encoded strings are found is split up character-by-character as stack strings. Each byte is moved one-by-one to a location on the stack before the decoding operation occurs. The first string of this type is in the function that resolves imports from kernel32.dll. This encoded string is shown in Figure 2.

```
000000014000165C | 48:81EC 48010000     | sub rsp,148                     | resolve_kernel32
0000000140001663 | C68424 F8000000 FB   | mov byte ptr ss:[rsp+F8],FB     |
000000014000166B | C68424 F9000000 FC   | mov byte ptr ss:[rsp+F9],FC     |
0000000140001673 | C68424 FA000000 FE   | mov byte ptr ss:[rsp+FA],FE     |
000000014000167B | C68424 FB000000 FF   | mov byte ptr ss:[rsp+FB],FF     |
0000000140001683 | C68424 FC000000 AA   | mov byte ptr ss:[rsp+FC],AA     |
000000014000168B | C68424 FD000000 44   | mov byte ptr ss:[rsp+FD],44     | 44:'D'
0000000140001693 | C68424 FE000000 3C   | mov byte ptr ss:[rsp+FE],3C     | 3C:'<'
000000014000169B | C68424 FF000000 5F   | mov byte ptr ss:[rsp+FF],5F     | 5F:'_'
00000001400016A3 | C68424 00010000 5B   | mov byte ptr ss:[rsp+100],5B    | 5B:'['
00000001400016AB | C68424 01010000 6E   | mov byte ptr ss:[rsp+101],6E    | 6E:'n'
00000001400016B3 | C68424 02010000 74   | mov byte ptr ss:[rsp+102],74    | 74:'t'
00000001400016BB | C68424 03010000 6B   | mov byte ptr ss:[rsp+103],6B    | 6B:'k'
00000001400016C3 | C68424 04010000 77   | mov byte ptr ss:[rsp+104],77    | 77:'w'
00000001400016CB | C68424 05010000 80   | mov byte ptr ss:[rsp+105],80    |
00000001400016D3 | C68424 06010000 7D   | mov byte ptr ss:[rsp+106],7D    | 7D:'}}'
00000001400016DB | C68424 07010000 67   | mov byte ptr ss:[rsp+107],67    | 67:'g'
00000001400016E3 | C68424 08010000 5F   | mov byte ptr ss:[rsp+108],5F    | 5F:'_'
00000001400016EB | C68424 09010000 86   | mov byte ptr ss:[rsp+109],86    |
00000001400016F3 | C68424 0A010000 81   | mov byte ptr ss:[rsp+10A],81    |
00000001400016FB | C68424 0B010000 83   | mov byte ptr ss:[rsp+10B],83    |
0000000140001703 | C68424 0C010000 75   | mov byte ptr ss:[rsp+10C],75    | 75:'u'
000000014000170B | C68424 0D010000 7E   | mov byte ptr ss:[rsp+10D],7E    | 7E:'~'
0000000140001713 | C68424 0E010000 45   | mov byte ptr ss:[rsp+10E],45    | 45:'E'
000000014000171B | C68424 0F010000 45   | mov byte ptr ss:[rsp+10F],45    | 45:'E'
0000000140001723 | C68424 10010000 70   | mov byte ptr ss:[rsp+110],70    | 70:'p'
000000014000172B | C68424 11010000 80   | mov byte ptr ss:[rsp+111],80    |
0000000140001733 | C68424 12010000 7B   | mov byte ptr ss:[rsp+112],7B    | 7B:'{'
000000014000173B | C68424 13010000 89   | mov byte ptr ss:[rsp+113],89    |
0000000140001743 | C68424 14010000 86   | mov byte ptr ss:[rsp+114],86    |
000000014000174B | C68424 15010000 7E   | mov byte ptr ss:[rsp+115],7E    | 7E:'~'
0000000140001753 | C68424 16010000 86   | mov byte ptr ss:[rsp+116],86    |
000000014000175B | C68424 17010000 4E   | mov byte ptr ss:[rsp+117],4E    | 4E:'N'
0000000140001763 | C68424 18010000 4E   | mov byte ptr ss:[rsp+118],4E    | 4E:'N'
000000014000176B | C68424 19010000 4B   | mov byte ptr ss:[rsp+119],4B    | 4B:'K'
0000000140001773 | C68424 1A010000 82   | mov byte ptr ss:[rsp+11A],82    |
000000014000177B | C68424 1B010000 8B   | mov byte ptr ss:[rsp+11B],8B    |
0000000140001783 | C68424 1C010000 8C   | mov byte ptr ss:[rsp+11C],8C    |
000000014000178B | C68424 1D010000 00   | mov byte ptr ss:[rsp+11D],0     |
0000000140001793 | C68424 1E010000 07   | mov byte ptr ss:[rsp+11E],7     |
000000014000179B | C68424 1F010000 6D   | mov byte ptr ss:[rsp+11F],6D    | 6D:'m'
00000001400017A3 | C68424 20010000 4F   | mov byte ptr ss:[rsp+120],4F    | 4F:'O'
00000001400017AB | C68424 21010000 72   | mov byte ptr ss:[rsp+121],72    | 72:'r'
00000001400017B3 | C68424 22010000 78   | mov byte ptr ss:[rsp+122],78    | 78:'x'
00000001400017BB | C68424 23010000 78   | mov byte ptr ss:[rsp+123],78    | 78:'x'
00000001400017C3 | C68424 24010000 78   | mov byte ptr ss:[rsp+124],78    | 78:'x'
00000001400017CB | C68424 25010000 78   | mov byte ptr ss:[rsp+125],78    | 78:'x'
00000001400017D3 | C68424 26010000 00   | mov byte ptr ss:[rsp+126],0     |
00000001400017DB | 48:8D8C24 F8000000   | lea rcx,qword ptr ss:[rsp+F8]   |
00000001400017E3 | E8 C44C0000          | call <68af2.decode_string>      |
00000001400017E8 | 48:8BC8              | mov rcx,rax                     |
00000001400017EB | E8 20340000          | call 68af2.140004C10            |
```

Figure 2: First Encoded String

This first string decodes to "C:\Windows\System32\kernel32.dll". This path is then used to read the DLL from the filesystem into memory. Imports are then resolved against this copy of the DLL rather than the system DLL. The function calls to copy the DLL are shown in Figure 3.

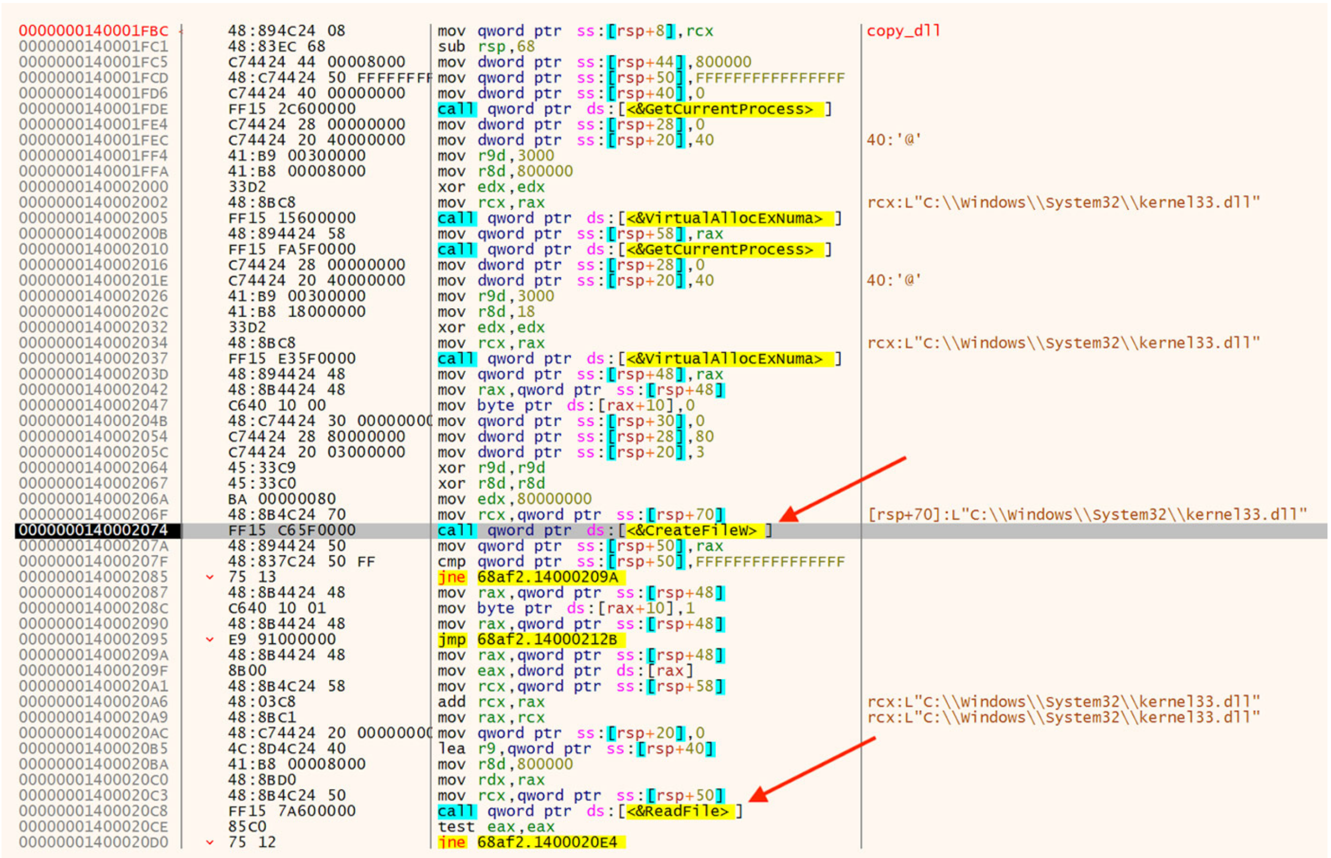

Figure 3: Copy DLL Function

In the debugger's environment, this read fails with an exception which then prevents the imports from being properly resolved [8]. A detailed explanation of what's happening here can be found on OALabs YouTube channel [9].To circumvent this trick, one can create a copy of the DLLs on the filesystem and change their names along with the decoded path strings. This way the DLLs can be read correctly and the imports properly resolved. This change can be done on the fly in the debugger after the strings are decoded. An example of changing this on the fly using the filename kernel33.dll is shown in Figure 4.

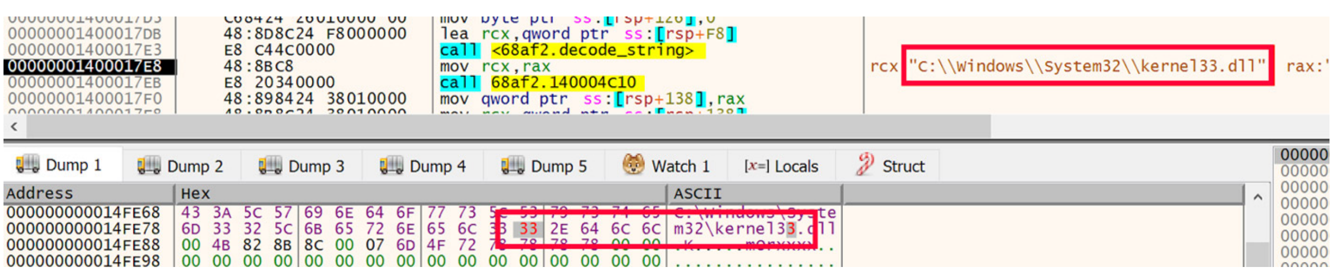

Figure 4: Change DLL Name

Alternatively, the single encoded byte needed for this change can be modified in the sample with a hex editor to make this change permanent. This also makes restarting the analysis in the debugger less annoying. This byte difference is highlighted using HexFiend's [10] comparison function and can be seen in Figure 5.

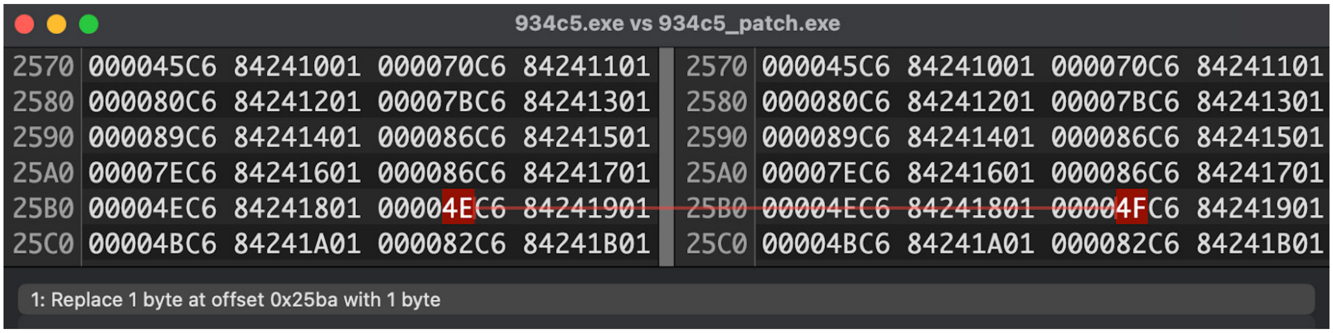

Figure 5: Original Compared to Patch

After the DLL path has been decoded, the various imports from that DLL are then decoded from similar stack strings. One example with LoadLibraryW is shown in Figure 6. Again, please note the trailing "mOrxxxx" string immediately after the encoded bytes of the "LoadLibraryW" string.

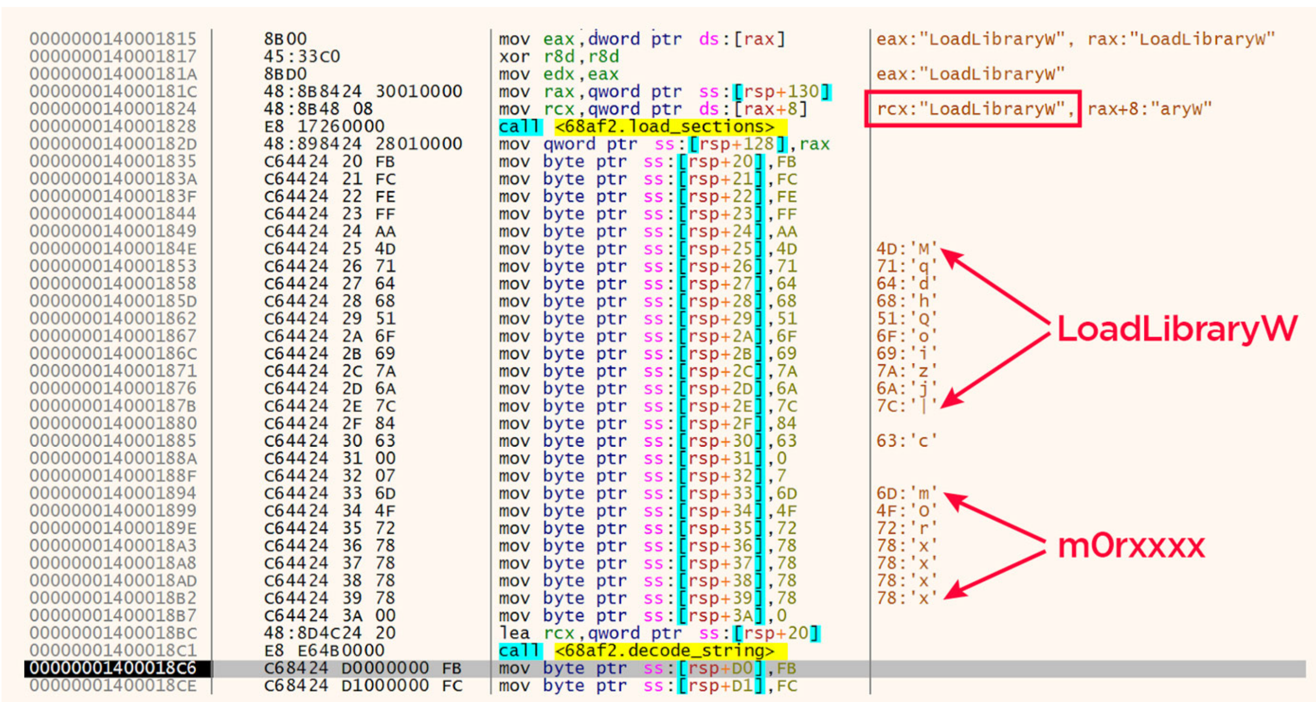

Figure 6: Encoded LoadLibraryW with Trailing Additional String

Once the import strings are decoded, a custom implementation of GetProcAddress is used on the copy of kernel32.dll to resolve the imports. The results of this process can be seen in Figure 7.

```
0000000140001D44 | 48:8D4C24 40          | lea rcx,qword ptr ss:[rsp+40]
0000000140001D49 | E8 5E470000           | call <68af2.decode_string>
0000000140001D4E | 48:8D5424 20          | lea rdx,qword ptr ss:[rsp+20]
0000000140001D53 | 48:8B8C24 28010000    | mov rcx,qword ptr ss:[rsp+128]
0000000140001D5B | E8 04F6FFFF           | call <68af2.get_proc_address>
0000000140001D60 | 48:8905 998C0000      | mov qword ptr ds:[<&JMP.&LoadLibraryW> ],rax
0000000140001D67 | 48:8D9424 A8000000    | lea rdx,qword ptr ss:[rsp+A8]
0000000140001D6F | 48:8B8C24 28010000    | mov rcx,qword ptr ss:[rsp+128]
0000000140001D77 | E8 E8F5FFFF           | call <68af2.get_proc_address>
0000000140001D7C | 48:8905 D58C0000      | mov qword ptr ds:[<&JMP.&GlobalMemoryStatusEx> ],rax
0000000140001D83 | 48:8D9424 80000000    | lea rdx,qword ptr ss:[rsp+80]
0000000140001D8B | 48:8B8C24 28010000    | mov rcx,qword ptr ss:[rsp+128]
0000000140001D93 | E8 CCF5FFFF           | call <68af2.get_proc_address>
0000000140001D98 | 48:8905 A98C0000      | mov qword ptr ds:[<&JMP.&GetDiskFreeSpaceExA> ],rax
0000000140001D9F | 48:8D5424 60          | lea rdx,qword ptr ss:[rsp+60]
0000000140001DA4 | 48:8B8C24 28010000    | mov rcx,qword ptr ss:[rsp+128]
0000000140001DAC | E8 B3F5FFFF           | call <68af2.get_proc_address>
0000000140001DB1 | 48:8905 808C0000      | mov qword ptr ds:[<&JMP.&GetProductInfo> ],rax
0000000140001DB8 | 48:8D5424 40          | lea rdx,qword ptr ss:[rsp+40]
0000000140001DBD | 48:8B8C24 28010000    | mov rcx,qword ptr ss:[rsp+128]
0000000140001DC5 | E8 9AF5FFFF           | call <68af2.get_proc_address>
0000000140001DCA | 48:8905 D78C0000      | mov qword ptr ds:[<&JMP.&GetVersionExA> ],rax
0000000140001DD1 | B8 01000000           | mov eax,1
0000000140001DD6 | 48:81C4 48010000      | add rsp,148
0000000140001DDD | C3                    | ret
```

Figure 7: After Imports Resolved

Examining the exports for these samples shows a DLL name "Input.exe" as well as one exported function "bsearch". These exports are shown in Figure 8.

```
140008ad2  }
140008ad2  char __pe_Input_export_dll_name[0xa] = "Input.exe", 0
140008adc  char __export_name(bsearch)[0x8] = "bsearch", 0
140008ae4  struct Import_Directory_Table __import_directory_entries[0x4] =
140008ae4  {
```

Figure 8: Exports

This bsearch function is a version of the binary search algorithm [11]. It appears once in the samples as part of the custom GetProcAddress implementation. This function call is highlighted in Figure 9.

```
int64_t get_proc_address(int64_t* arg1, int64_t arg2)

1400015af  488b442430          mov   rax, qword [rsp+0x30 {var_58_1}]
1400015b4  8b4018              mov   eax, dword [rax+0x18]
1400015b7  488d0dba3b0000      lea   rcx, [rel sub_140005178]
1400015be  48894c2420          mov   qword [rsp+0x20 {var_68}], rcx  {sub_140005178}
1400015c3  41b910000000        mov   r9d, 0x10
1400015c9  448bc0              mov   r8d, eax
1400015cc  488b442450          mov   rax, qword [rsp+0x50 {var_38}]
1400015d1  488b5050            mov   rdx, qword [rax+0x50]
1400015d5  488d8c2498000000    lea   rcx, [rsp+0x98 {arg_10}]
1400015dd  e83a270000          call  bsearch
1400015e2  4889442470          mov   qword [rsp+0x70 {var_18_1}], rax
1400015e7  48837c247000        cmp   qword [rsp+0x70 {var_18_1}], 0x0
1400015ed  750f                jne   0x1400015fe
```

Figure 9: Function Call to bsearch in Custom GetProcAddress

After the imports from kernel32.dll have been resolved, the next DLL is ntdll.dll. The process shown above is repeated for this DLL. The alternative name used here was npdll.dll. The path after this change is shown in Figure 10.

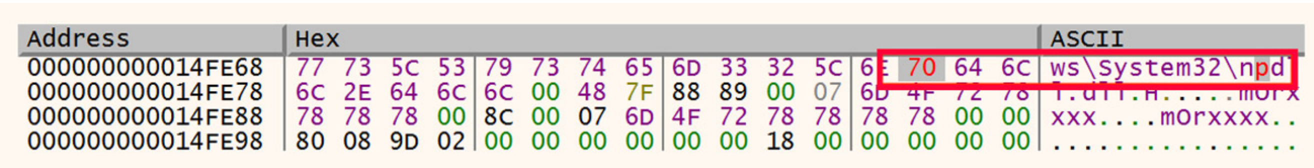

Figure 10: DLL Name Change

Some of the resolved imports give an idea of what's to come and what the capabilities are for these samples. Examples of this are the imports of "HttpAddRequestHeadersW" and "EnumProcesses" as shown in Figures 11 and 12.

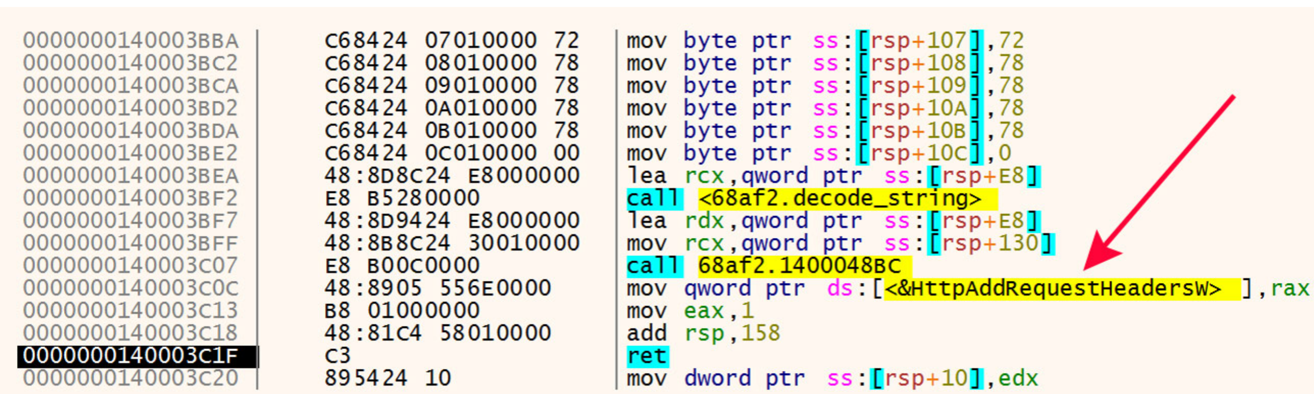

Figure 11: Import HttpAddRequestHeadersW

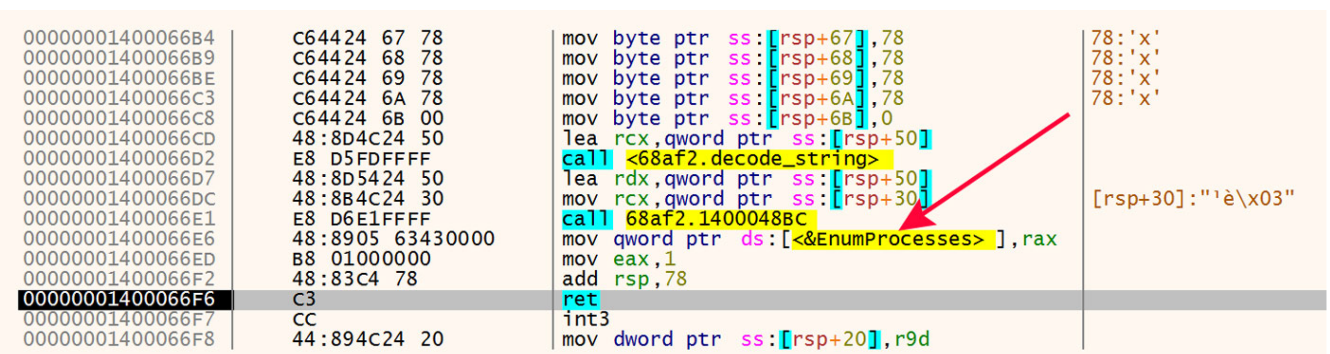

Figure 12: Import EnumProcesses

The rest of the DLLs after ntdll.dll that are loaded are not loaded using this same antianalysis trick. These other DLLs are user32.dll, wininet.dll, and psapi.dll. The steps to decode and resolve imports from each DLL are divided into separate functions. Each of these functions is shown in figure 13.

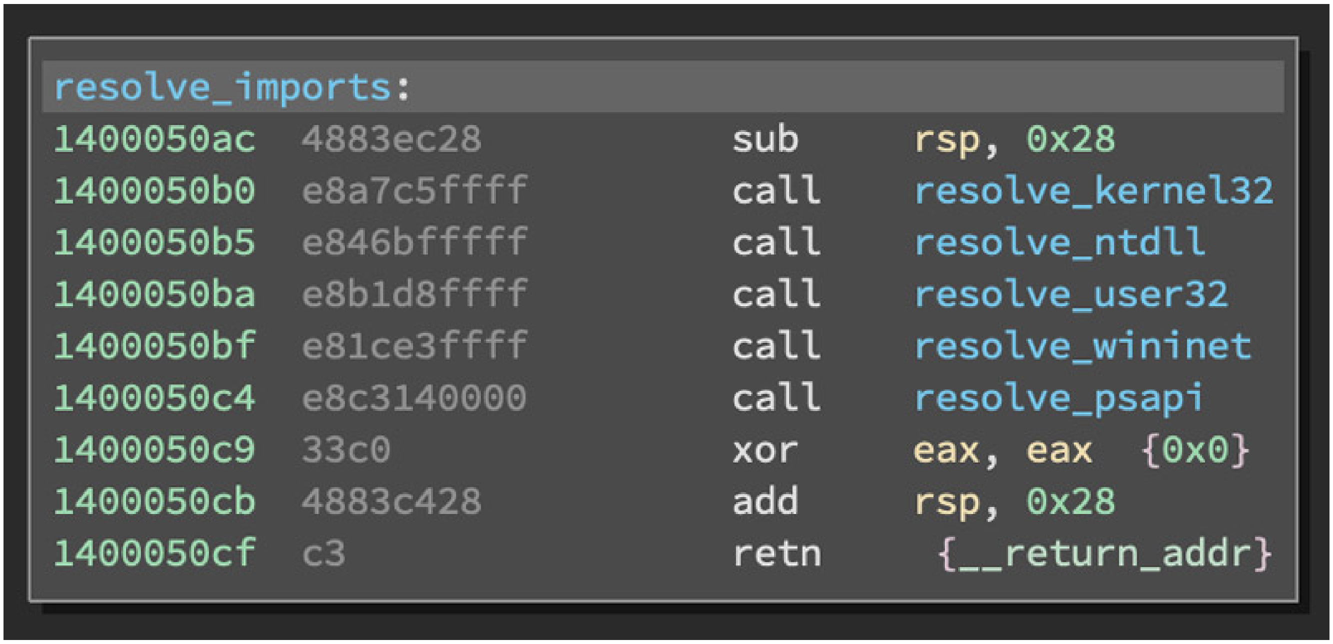

Figure 13: Resolve Import Functions

In the code of these samples, another interesting library function calling pattern is to call NtAllocateVirtualMemory using syscall to allocate memory. This pattern of function call is shown in Figure 14.

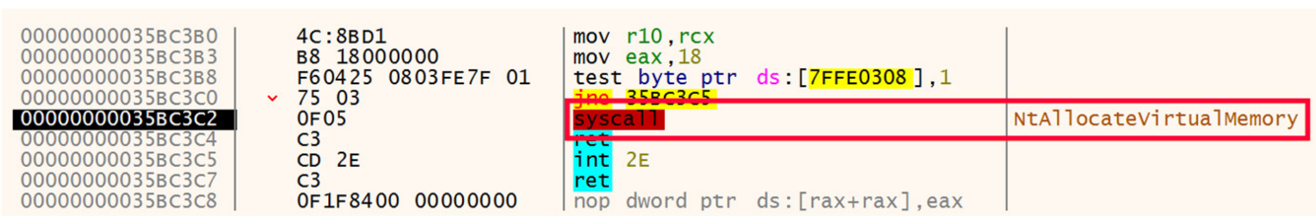

Figure 14: Syscall Used on NtAllocateVirtualMemory

Collect Environment Information

The first set of capabilities in these samples is to collect information about the victim's environment. The first bit of information collected is the name of the computer. The call to GetComputerNameExA is shown in Figure 15.

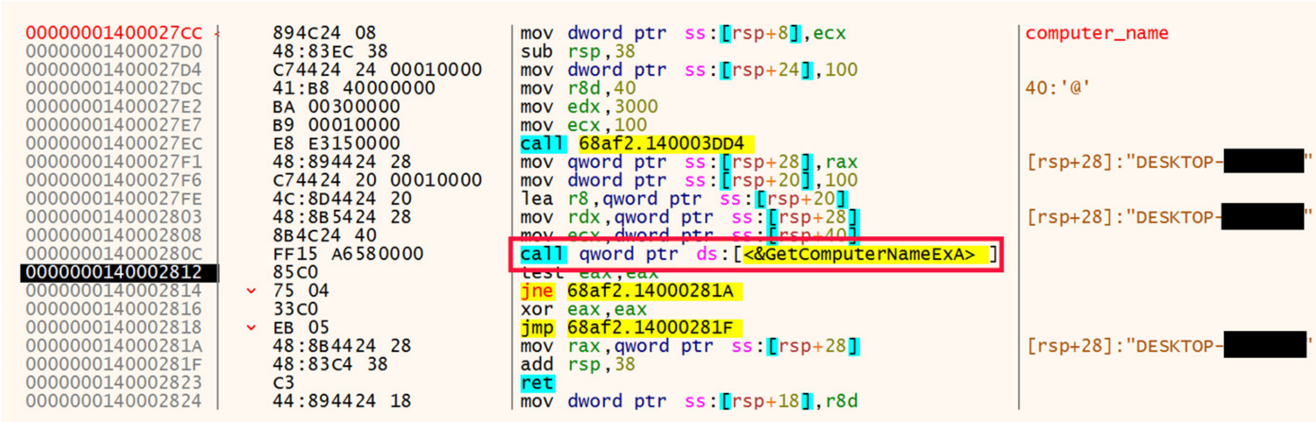

Figure 15: Collect Computer Name

The next bit of environmental information collected is the physical and virtual memory status. This is done via a call to GlobalMemoryStatusEx [12] which is shown in Figure 16.

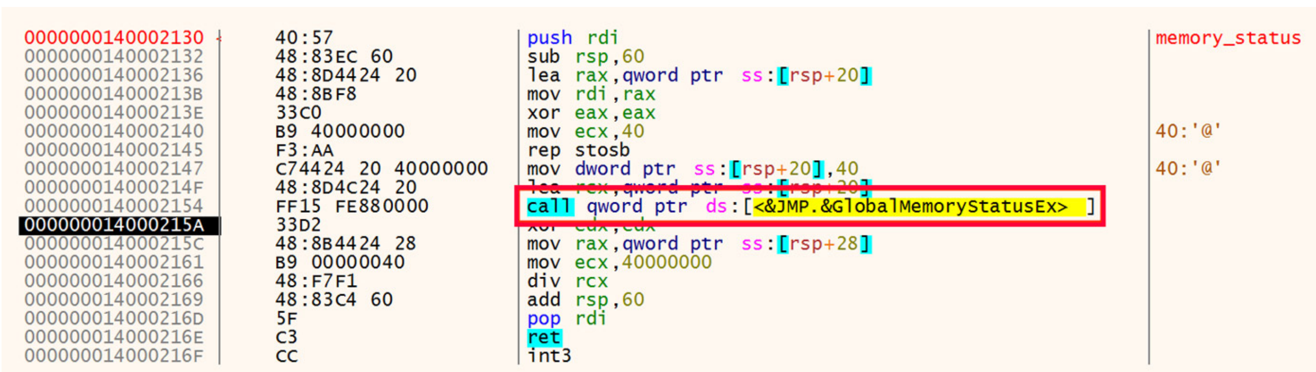

Figure 16: Measure Physical and Virtual Memory Status

The memory status is not sent back to the command and control (C2) infrastructure. It is probably used in the file processing algorithm because the primary purpose of these samples is to exfiltrate files from the victim's computer. These files must be copied from the filesystem to memory for processing before being sent to the C2.

The next data point collected is the username that ran the malware file. This data point does not appear to be sent back to the C2 according to the fields in the network traffic. The API call to GetUserNameA is shown in Figure 17.

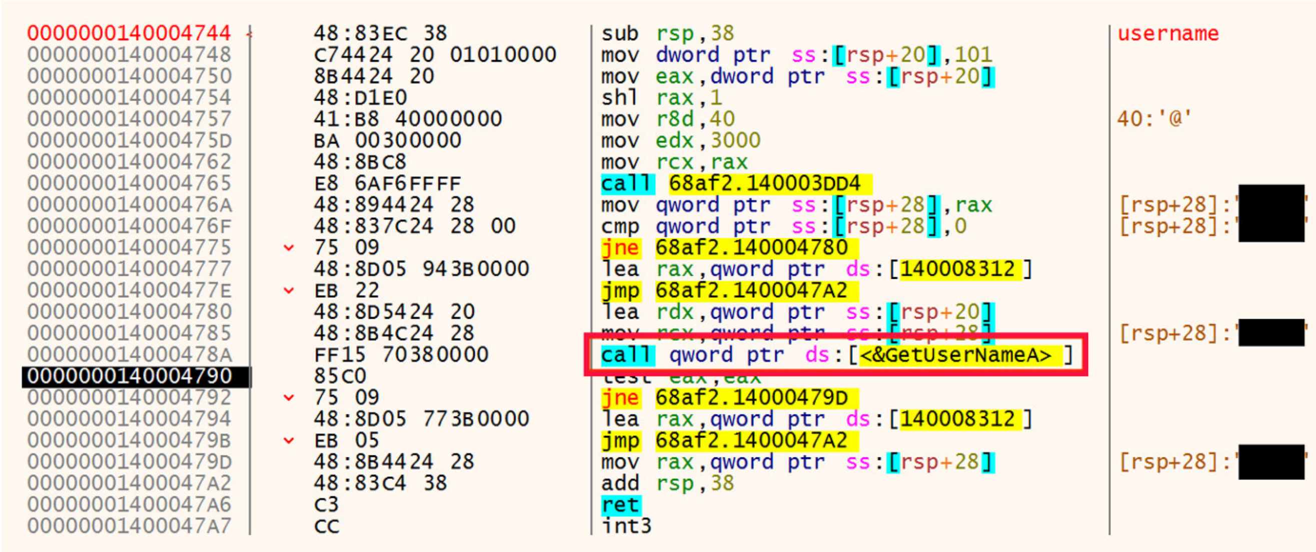

Figure 17: Collect Username

Next the samples check the free disk space via GetDiskFreeSpaceExA as shown in Figure 18.

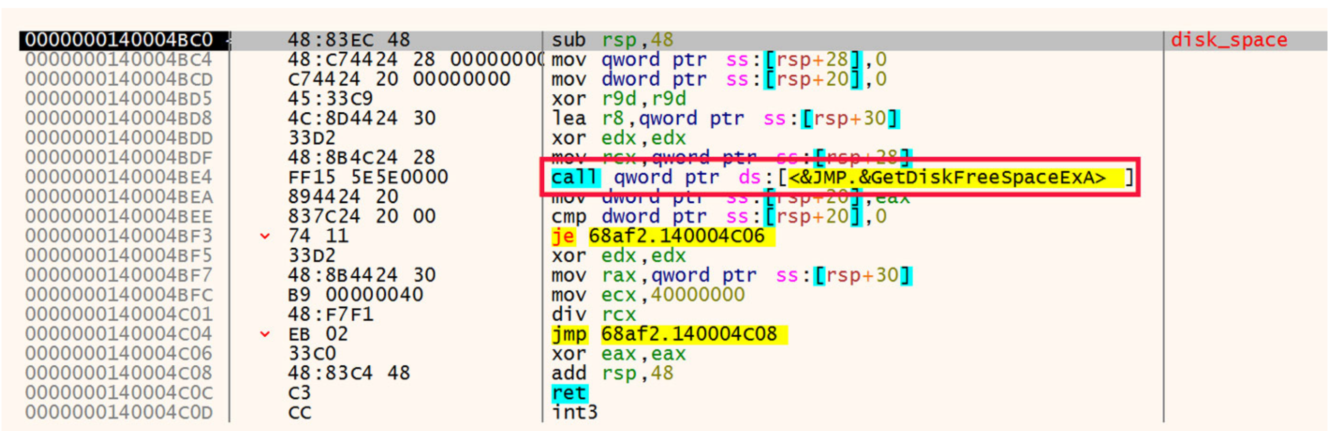

Figure 18: Check Disk Free Space

Next the OS version and product information is collected via calls to GetVersionExA and GetProductInfo. These calls are shown in Figure 19.

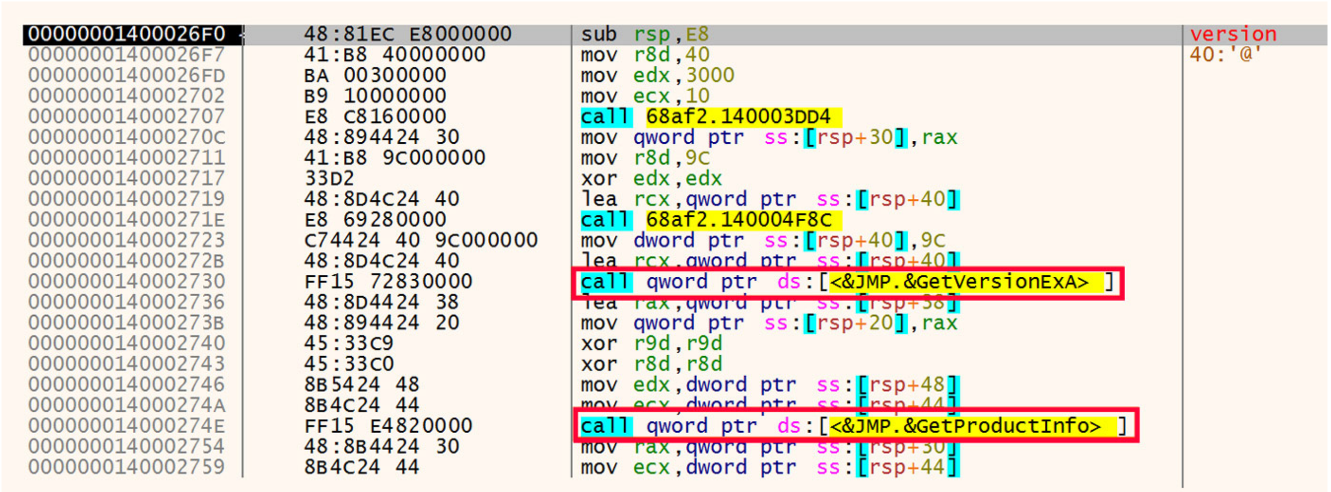

Figure 19: Gather OS Version and Product Information

Malware Configuration

After the environment information has been collected, the malware configuration strings are then decoded. As opposed to the stack strings used in the import resolution process, the configuration strings are normal strings in the .data section as shown above. All of these strings have a trailing "mOrxxxx" string. Interestingly this additional data does not cause problems for the decoding process. The reason for this is the decoding function works on a null terminated string. Examining the encoded strings closely, this null termination can be seen before the additional characters. An example of this null termination in a configuration string is shown in Figure 20.

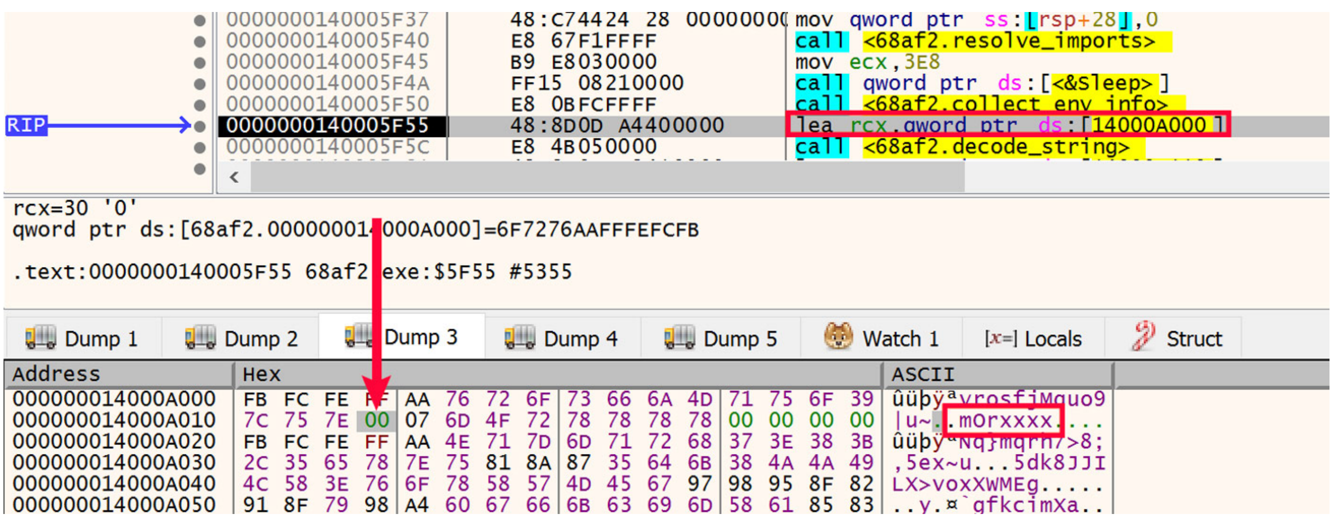

Figure 20: Null Termination

An example of this null termination in a stack string is shown in Figure 21.

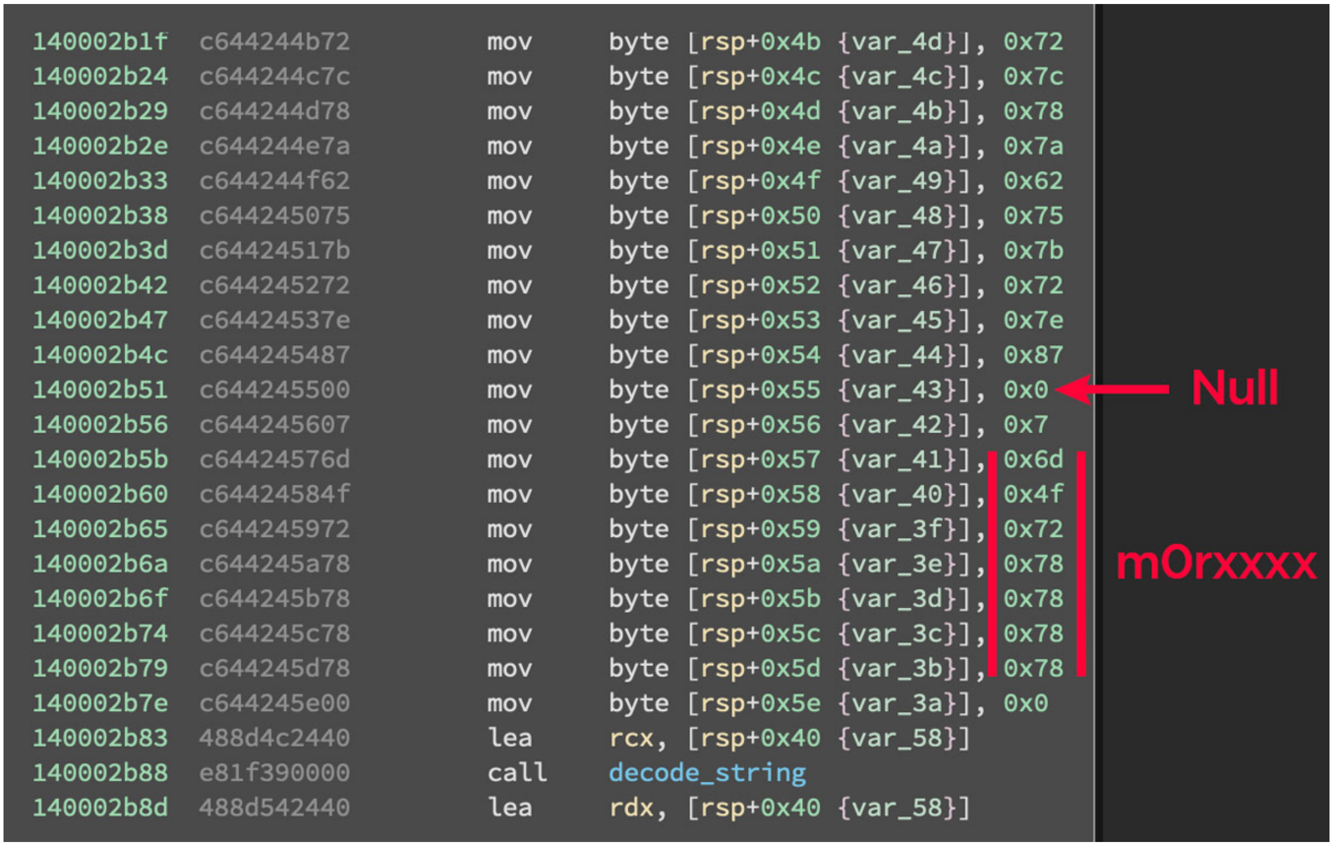

Figure 21: Null Termination in Stack String

The configuration strings for one particular sample [13] are shown in Figure 22.

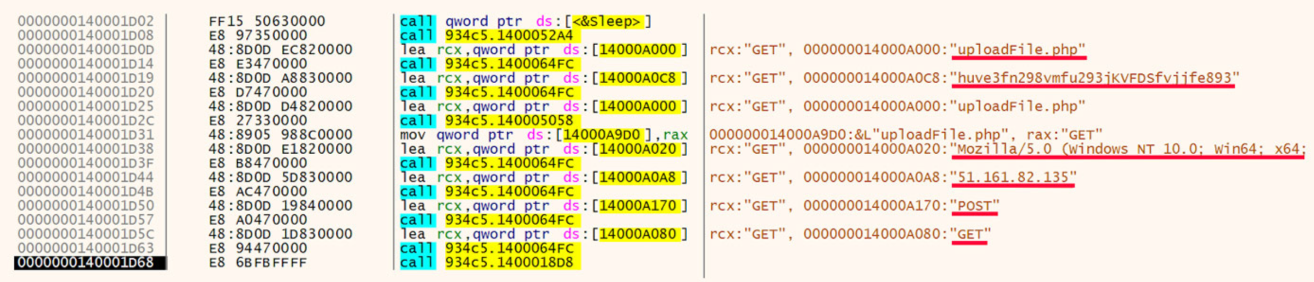

Figure 22: Decoded Configuration Strings

The second configuration string from the top in Figure 22 above is used in a field called "key" in the C2 traffic along with the exfiltrated data. Each of the samples analyzed here have different key strings. The following table shows each of these strings along with the first five characters of the SHA256 hash of the file the string was collected from.

SHA256 Prefix	Key
dcc4a	46rnyegq235etnerhgf43trrthgbfRYdfnhg
68af2	8953n7b8ewurdfb3njnyuridrwdb
934c5	huve3fn298vmfu293jKVFDSfvjjfe893

a7cf0	huve3fn298vmfu293jKVFDSfvjjfe893
8cfd5	3f9n8uv0n43809vn3d092v09290

Exfiltration

The data exfiltration process starts by enumerating the logical drives that are available on the victim's computer. This is determined using a call to GetLogicalDriveStringsW and is shown in Figure 23.

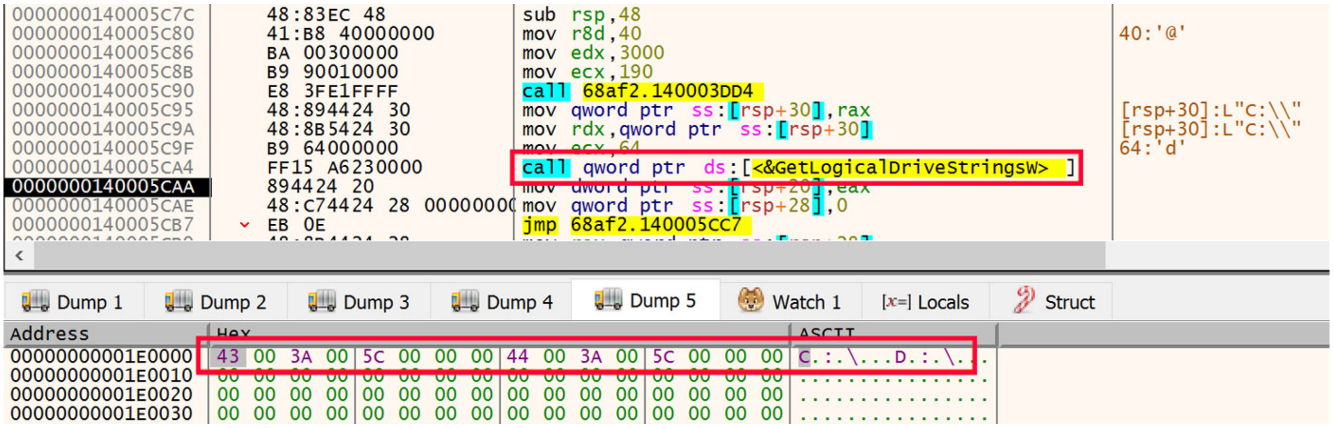

Figure 23: Determine Available Logical Drives

For each of these logical drives, a function is called that walks the file system searching for targeted files and exfiltrating them. This walk function interestingly is recursive. This recursion is shown in Figure 24.

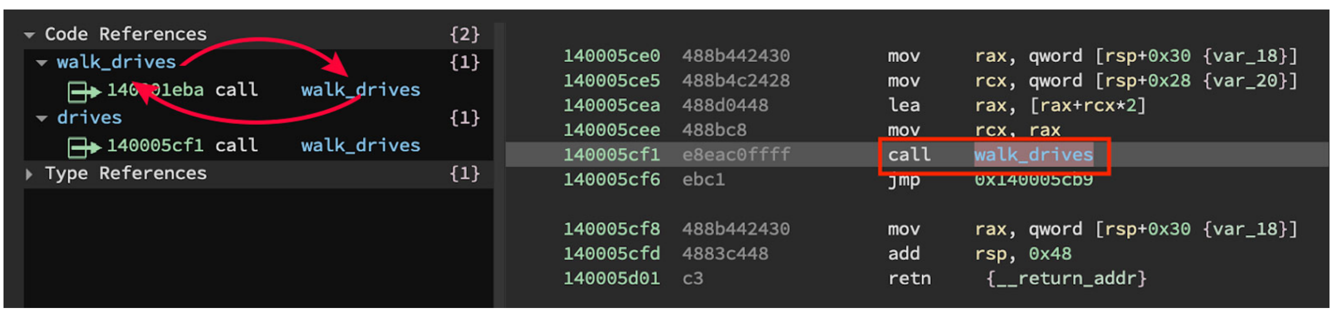

Figure 24: Walk Function Recursion

The first steps taken in the walk drives function is to add an asterisk to the path that is the input of the function. This is numbered "1" below. Then memory is allocated twice in a row. This is numbered "2" below. The string with the trailing asterisk is then written to one of the two allocated memory locations. This is numbered "3" below. Finally, this string is used to call "FindFirstFileW" with the second allocated memory location as the output location that receives the structure resulting from the API call. These are all shown in Figure 25.

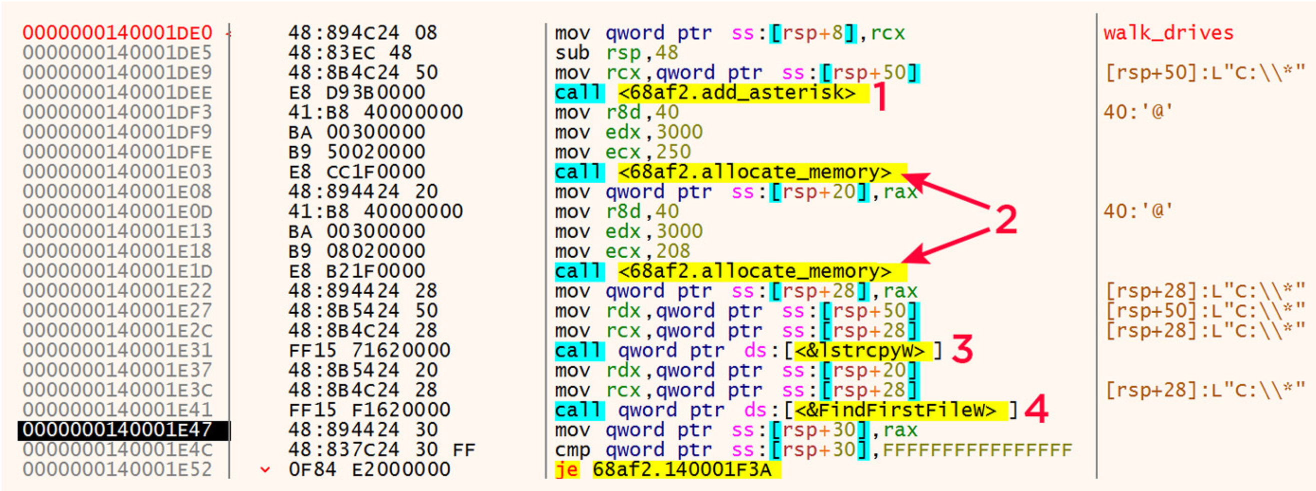

Figure 25: Finding Files

As the malware walks the file system, any files that contain one of the following strings in the filename are exfiltrated.

.doc	.xls	.pdf
.docx	.xlsx	

Interestingly, the algorithm used to find these files is probably not what the adversary expected. Rather than checking for a file extension as a suffix it actually matches any infix of the above strings. Because of this, any file with .xlsx will already match .xls for example. These target file extensions are shown in Figure 26.

Figure 26: Target File Extensions

Next the file size is determined using a call to GetFileSize. This information is included in the exfiltrated data. The API call is shown in Figure 27.

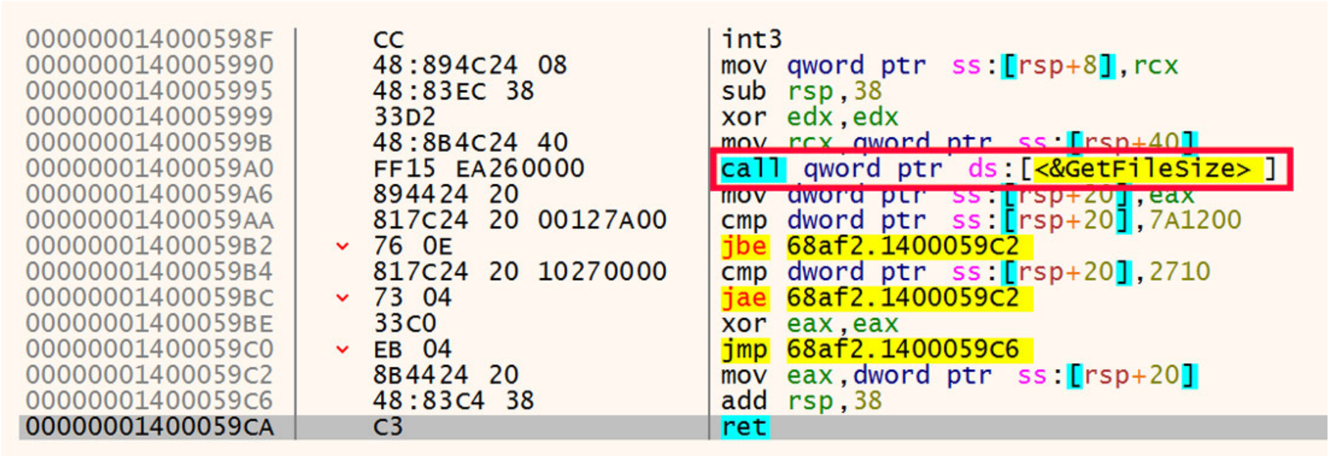

Figure 27: Determine File Size

As the file system walk proceeds, the path strings are emitted as debug strings via a call to OutputDebugStringW. This is followed immediately by bytes for a Windows carriage return line feed. These two are shown in Figure 28.

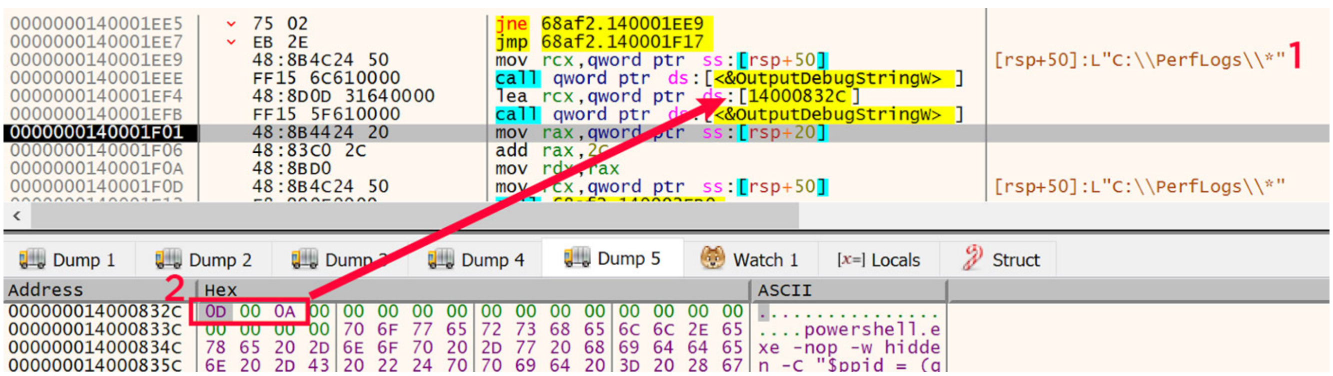

Figure 28: Emit Debug Strings

This malware can exfiltrate large files. It does this by dividing the file into chunks according to a hard coded "frame size". This hard coded size is 32535 bytes and is highlighted in Figure 29.

```
1400042f8  4c8d05115e0000      lea   r8, [rel data_14000a110]
1400042ff  488d15a63f0000      lea   rdx, [rel data_1400082ac]
140004306  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
14000430b  e8581a0000          call  sub_140005d68
140004310  4c8b442430          mov   r8, qword [rsp+0x30 {var_48}]
140004315  488d159c3f0000      lea   rdx, [rel data_1400082b8]  {"data"}
14000431c  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
140004321  e8421a0000          call  sub_140005d68
140004326  41b80a000000        mov   r8d, 0xa
14000432c  488b542428          mov   rdx, qword [rsp+0x28 {var_50}]
140004331  488b842488000000    mov   rax, qword [rsp+0x88 {arg_10}]
140004339  488b4818            mov   rcx, qword [rax+0x18]
14000433d  e8e2e4ffff          call  sub_140002824
140004342  4c8b442428          mov   r8, qword [rsp+0x28 {var_50}]
140004347  488d15723f0000      lea   rdx, [rel data_1400082c0]  {"filesize"}
14000434e  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
140004353  e8101a0000          call  sub_140005d68
140004358  41b80a000000        mov   r8d, 0xa
14000435e  488b542428          mov   rdx, qword [rsp+0x28 {var_50}]
140004363  b9177f0000          mov   ecx, 0x7f17          32535
140004368  e8b7e4ffff          call  sub_140002824
14000436d  4c8b442428          mov   r8, qword [rsp+0x28 {var_50}]
140004372  488d15573f0000      lea   rdx, [rel data_1400082d0]  {"framesize"}
140004379  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
14000437e  e8e5190000          call  sub_140005d68
140004383  8b842490000000      mov   eax, dword [rsp+0x90 {arg_18}]
14000438a  41b80a000000        mov   r8d, 0xa
140004390  488b542428          mov   rdx, qword [rsp+0x28 {var_50}]
140004395  8bc8                mov   ecx, eax
140004397  e888e4ffff          call  sub_140002824
14000439c  4c8b442428          mov   r8, qword [rsp+0x28 {var_50}]
1400043a1  488d15383f0000      lea   rdx, [rel data_1400082e0]  {"framenum"}
1400043a8  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
1400043ad  e8b6190000          call  sub_140005d68
1400043b2  41b80a000000        mov   r8d, 0xa
1400043b8  488b542428          mov   rdx, qword [rsp+0x28 {var_50}]
1400043bd  488b842488000000    mov   rax, qword [rsp+0x88 {arg_10}]
1400043c5  488b4808            mov   rcx, qword [rax+0x8]
1400043c9  e856e4ffff          call  sub_140002824
1400043ce  4c8b442428          mov   r8, qword [rsp+0x28 {var_50}]
1400043d3  488d15163f0000      lea   rdx, [rel data_1400082f0]  {"filecrc"}
1400043da  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
1400043df  e884190000          call  sub_140005d68
1400043e4  488b842488000000    mov   rax, qword [rsp+0x88 {arg_10}]
1400043ec  4c8b00              mov   r8, qword [rax]
1400043ef  488d15023f0000      lea   rdx, [rel data_1400082f8]  {"filename"}
1400043f6  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
1400043fb  e868190000          call  sub_140005d68
140004400  4c8b05c9660000      mov   r8, qword [rel data_14000aad0]
140004407  488d15a23e0000      lea   rdx, [rel data_1400082b0]  {"pcname"}
14000440e  488b4c2420          mov   rcx, qword [rsp+0x20 {var_58}]
140004413  e850190000          call  sub_140005d68
```

Figure 29: Frame Size

The above also shows all the other fields that are used in the C2 traffic. The frame number starts at "-0" then "1", "2" etc. The "filecrc" field is the cyclic redundancy check (CRC) which is used as a checksum for error detection [14]. The last two fields are the filename and the computer name.

Using a constructed test file named "testfile.doc" the TLS encrypted C2 traffic is intercepted and analyzed using Burp [15] and Inetsim [16]. The body parameters and the request headers from this test file are shown in Figure 30.

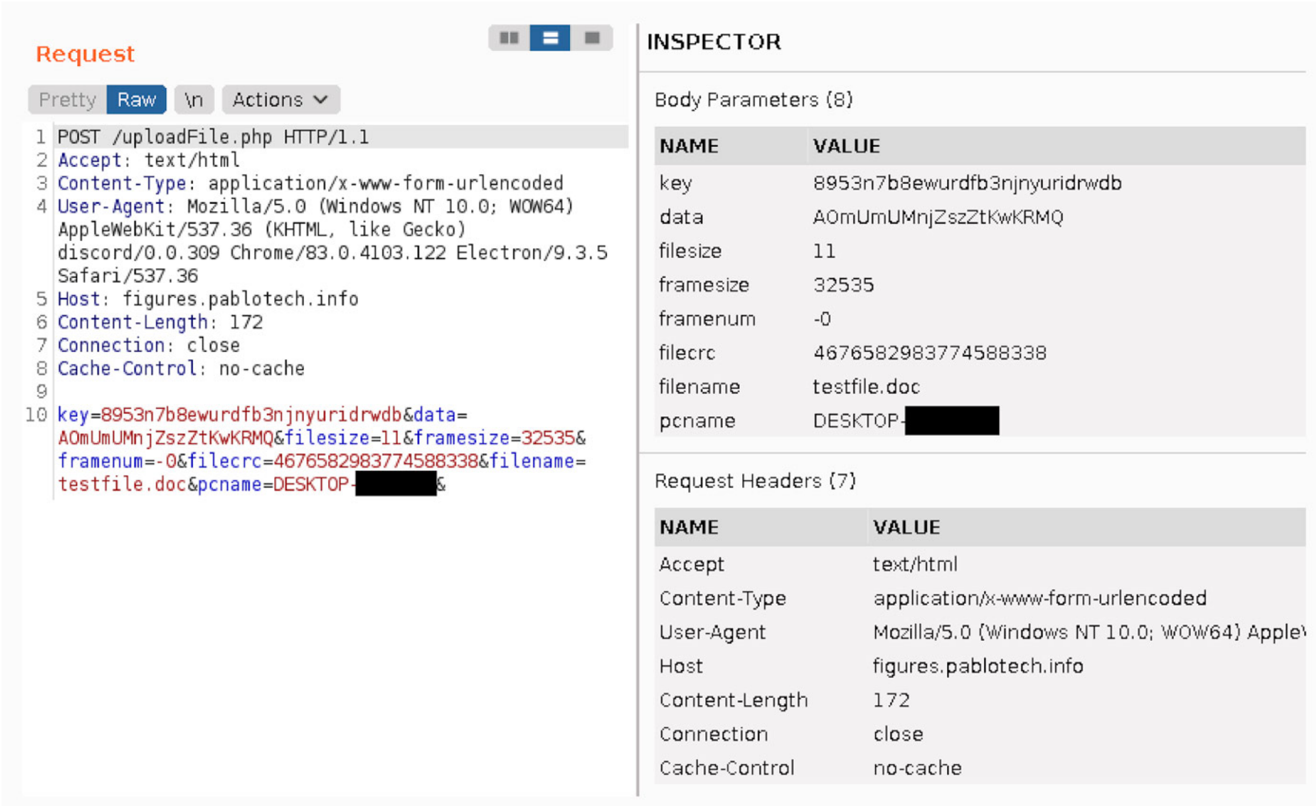

Figure 30: Test Document Shown Being Exfiltrated in C2 Network Traffic

Interestingly, the configuration strings include "GET" in addition to "POST". However, this capability does not appear to be used in these samples. A check is performed which determines which of the two request methods are used. This check is shown at the top of Figure 31. The POST and GET options are shown in the center, and the call to HttpOpenRequestW is shown at the bottom.

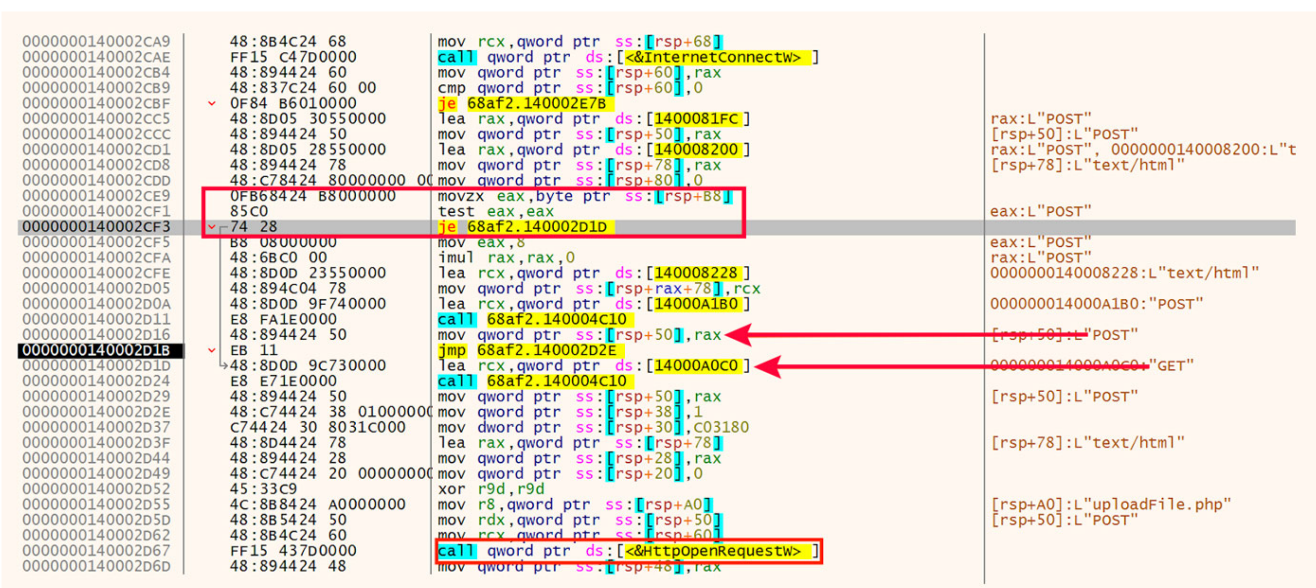

Figure 31: Request Method Options

The algorithm used to determine which files are exfiltrated is flawed in that it will match files that are not Word, Excel, or PDF documents. It will exfiltrate any file that contains the target strings anywhere in the filename. A test of this is shown in Figure 32 which uses a fake file with the extension ".txt" and ".pdf" in the middle of the filename.

Body Parameters (8)

NAME	VALUE
key	8953n7b8ewurdfb3njnyuridrwdb
data	AOmUmUMnIA==
filesize	4
framesize	32535
framenum	-0
filecrc	-)
filename	this_is_not.pdf-it-is-text.txt
pcname	DESKTOP-[redacted]

Figure 32: Unexpected Exfiltration

After all the filesystem walking has completed, the next function called sends a dummy "end of transmission" file out to the C2. The filename of the file is ".lock" and the content is "locked". This file only exists in memory and network traffic. It is not written to the filesystem. The strings for this dummy file are shown in Figure 33.

```
sub_1400044b8:
1400044b8  4056               push    rsi {__saved_rsi}
1400044ba  57                 push    rdi {__saved_rdi}
1400044bb  4881ecf8000000     sub     rsp, 0xf8
1400044c2  c744242000000000   mov     dword [rsp+0x20 {var_e8}], 0x0
1400044ca  488d05333e0000     lea     rax, [rel data_140008304]  {"locked"}
1400044d1  4889442468         mov     qword [rsp+0x68 {var_a0}], rax  {data_140008304, "locked"}
1400044d6  48c744247007000…   mov     qword [rsp+0x70 {var_98}], 0x7
1400044df  48c7442438010000…  mov     qword [rsp+0x38 {var_d0}], 0x1
1400044e8  48c7442448070000…  mov     qword [rsp+0x48 {var_c0}], 0x7
1400044f1  48c7442440010000…  mov     qword [rsp+0x40 {var_c8}], 0x1
1400044fa  488d050b3e0000     lea     rax, [rel data_14000830c]  {".lock"}
140004501  4889442430         mov     qword [rsp+0x30 {var_d8}], rax  {data_14000830c, ".lock"}
140004506  488d8424a0000000   lea     rax, [rsp+0xa0 {var_68}]
14000450e  488d4c2430         lea     rcx, [rsp+0x30 {var_d8}]
```

Figure 33: Dummy File Strings

This dummy file as seen in the network traffic is shown in Figure 34.

Body Parameters (8)

NAME	VALUE
key	8953n7b8ewurdfb3njnyuridrwdb
data	locked
filesize	7
framesize	32535
framenum	-0
filecrc	1
filename	.lock
pcname	DESKTOP-[redacted]

Figure 34: End of Transmission Dummy File

After all of the above is finished, the last action taken by the samples is to run a PowerShell command from a string. This command gets the process ID of the parent process of the command itself. It uses that process ID to kill the malware's process. It then deletes the malware file from the filesystem. This is to clean up after the data exfiltration is complete. The command is executed by a call to CreateProcessA as shown in Figure 35.

```
sub_1400050d0:
1400050d0  4881ece8000000     sub   rsp, 0xe8
1400050d7  41b868000000       mov   r8d, 0x68
1400050dd  33d2               xor   edx, edx  {0x0}
1400050df  488d4c2470         lea   rcx, [rsp+0x70 {var_78}]
1400050e4  e8a3feffff         call  sub_140004f8c
1400050e9  41b818000000       mov   r8d, 0x18
1400050ef  33d2               xor   edx, edx  {0x0}
1400050f1  488d4c2458         lea   rcx, [rsp+0x58 {var_90}]
1400050f6  e891feffff         call  sub_140004f8c
1400050fb  c744247068000000   mov   dword [rsp+0x70 {var_78}], 0x68
140005103  488d0536320000     lea   rax, [rel data_140008340]  {"powershell.exe -nop -w hidden -C…"}
14000510a  4889442450         mov   qword [rsp+0x50 {var_98}], rax  {data_140008340, "powershell.exe -nop -w hidden -C…"  }
14000510f  488d442458         lea   rax, [rsp+0x58 {var_90}]
140005114  4889442448         mov   qword [rsp+0x48 {var_a0}], rax {var_90}
140005119  488d442470         lea   rax, [rsp+0x70 {var_78}]
14000511e  4889442440         mov   qword [rsp+0x40 {var_a8}], rax {var_78}
140005123  48c7442438000000…  mov   qword [rsp+0x38 {var_b0}], 0x0
14000512c  48c7442430000000…  mov   qword [rsp+0x30 {var_b8}], 0x0
140005135  c744242800000000   mov   dword [rsp+0x28 {var_c0}], 0x0
14000513d  c744242000000000   mov   dword [rsp+0x20 {var_c8}], 0x0
140005145  4533c9             xor   r9d, r9d  {0x0}
140005148  4533c0             xor   r8d, r8d  {0x0}
14000514b  488b542450         mov   rdx, qword [rsp+0x50 {var_98}]  {data_140008340, "powershell.exe -nop -w hidden -C…"  }
140005150  33c9               xor   ecx, ecx  {0x0}
140005152  ff15582f0000       call  qword [rel CreateProcessA@IAT]
140005158  baffffffff         mov   edx, 0xffffffff
14000515d  488b4c2458         mov   rcx, qword [rsp+0x58 {var_90}]
140005162  ff15d02e0000       call  qword [rel WaitForSingleObject@IAT]
140005168  4881c4e8000000     add   rsp, 0xe8
14000516f  c3                 retn   {__return_addr}
```

Figure 35: PowerShell Cleanup Command

The full text of the PowerShell command is shown in Figure 36.

```
powershell.exe -nop -w hidden -C "$ppid = (gwmi win32_process | ? processid -eq
$PID).parentprocessid; $proc = Get-Process -FileVersionInfo -Id $ppid; Stop-Process -Force
-ErrorAction SilentlyContinue -Id $ppid; $buff = [byte[]]@(, 0 * 1mb); Set-Content -Path
$proc.FileName -Force -Confirm:0 -Value $buff; Remove-Item -Path $proc.FileName -Force
-Confirm:0 "
```

PowerShell Command

Evolving Variants

The earliest observed variant of this malware family was compiled on April 24th, 2021 according to the PE header TimeDateStamp field. It was then first seen in the Titanium Platform on April 25th, 2021. This earliest variant did not include the PowerShell cleanup that was used in the later two variants. The comparison of the older sample [18] and the newer sample [19] analyzed using Relyze [20] is shown in Figure 36.

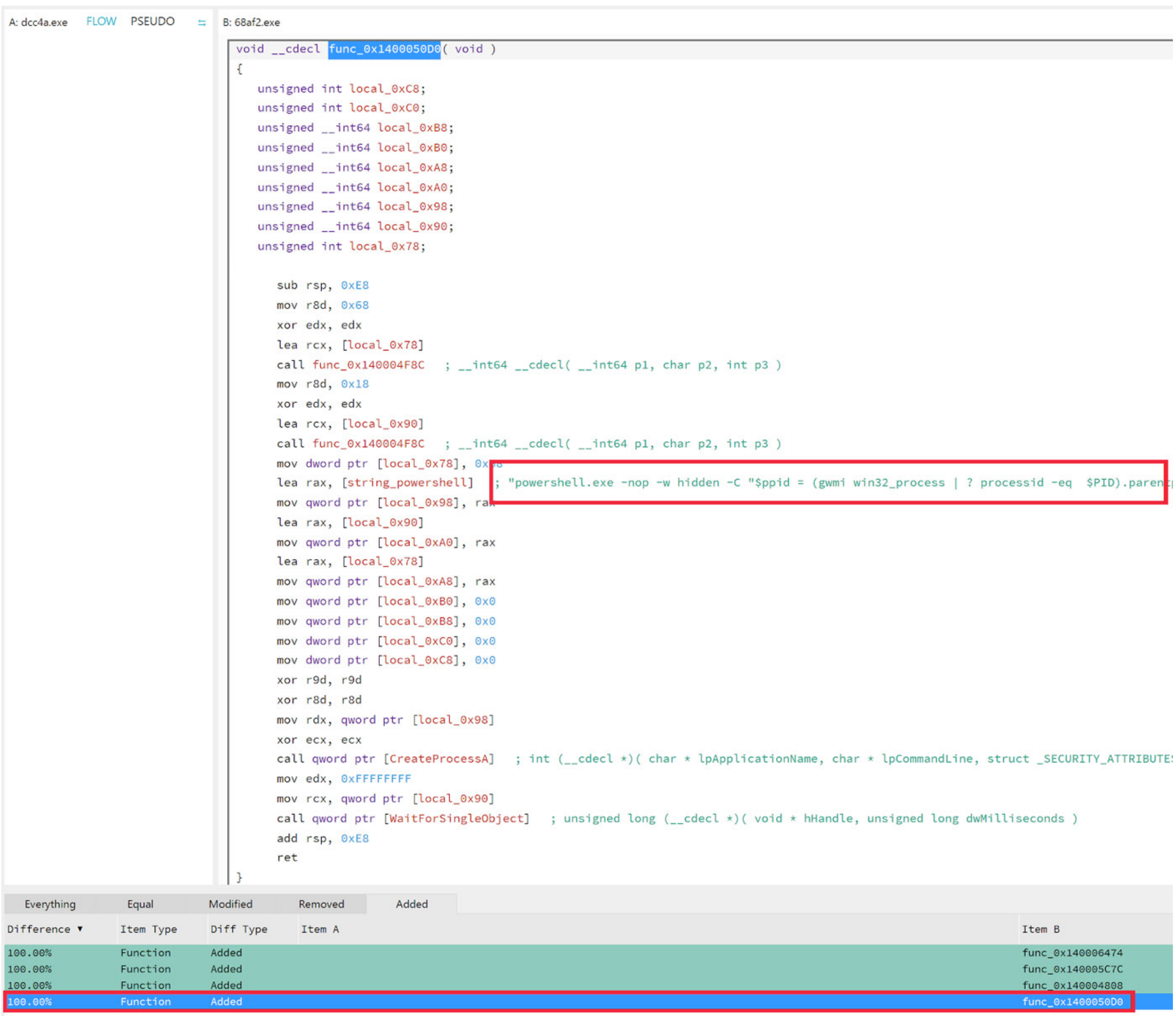

Figure 36: Addition of PowerShell Cleanup Capability

Another difference between this oldest sample and most of the newer ones is the inclusion of a program database (PDB) path. This path is shown in Figure 37.

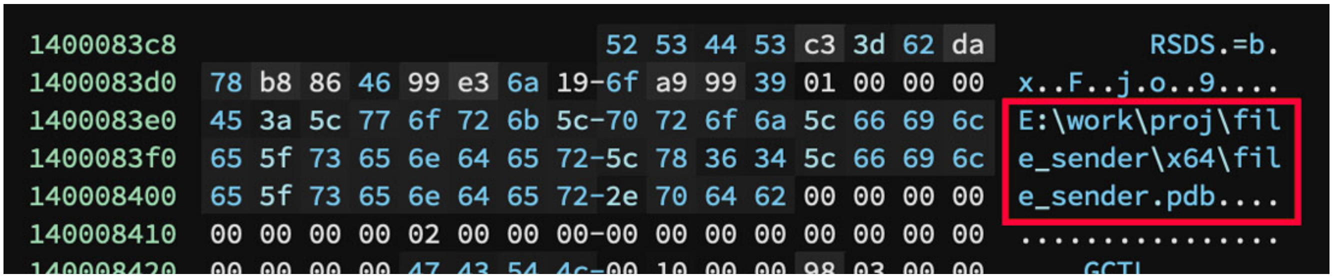

Figure 37: PDB Path String

The two newer samples have very few differences. Code-wise, there is one single function that is ~55% different between the two. The rest of the code is identical or nearly identical. This small difference is shown in Figure 38.

Figure 38: Difference in Newer Samples

Another interesting difference between the two newer samples is that the newest sample does not call back to a hostname for C2 communication. It is configured to call back to the C2 URL on a bare IP address.

Related Malware

One function in particular that is found in each of the three samples analyzed above is the string decoding function. Encoding and decoding functions are a good area of code to examine closely and to hunt for in malware repositories. This type of function can be stable across samples and tends to be reused by an adversary in multiple campaigns even if the capabilities of the malware are radically different. Building a YARA rule based on this function reveals an older malware sample [21] which was blogged about by researchers at Walmart [22]. The results of a retrohunt in the Titanium Platform using this YARA rule is shown in Figure 39. The result shown in orange is a false positive. This file is a copy of one of the other actual malware files but appears to have been modified by a researcher.

Figure 39: Retrohunt Results

This sample is definitely related to the three samples analyzed here. It uses the same anti-analysis trick as well as the same string obfuscation including the "mOrxxxx" trailing characters. Both the standard string form as well as the stack strings are found in this sample. However, this older sample is a CobaltStrike beacon loader rather than a data exfiltrator. The YARA rule for detecting this code overlap is provided at the end of the blog. As opposed to earlier YARA rules that I have written, I read the advice from Marc Ochsenmeier on Twitter about adding comments with the meaning of opcodes in YARA rules meant for sharing [23]. This rule and future rules will include the assembly as well as the bytecode strings.

In addition to this older malware sample that is definitely related, a hunt for other malware that contains variations of the string "mOrxxxx" reveals a multitude of potentially related malware samples. These are nearly uniformly detected as malicious. A future blog will address these additional files. The results of a retrohunt for the full string "mOrxxxx" is shown in Figure 40.

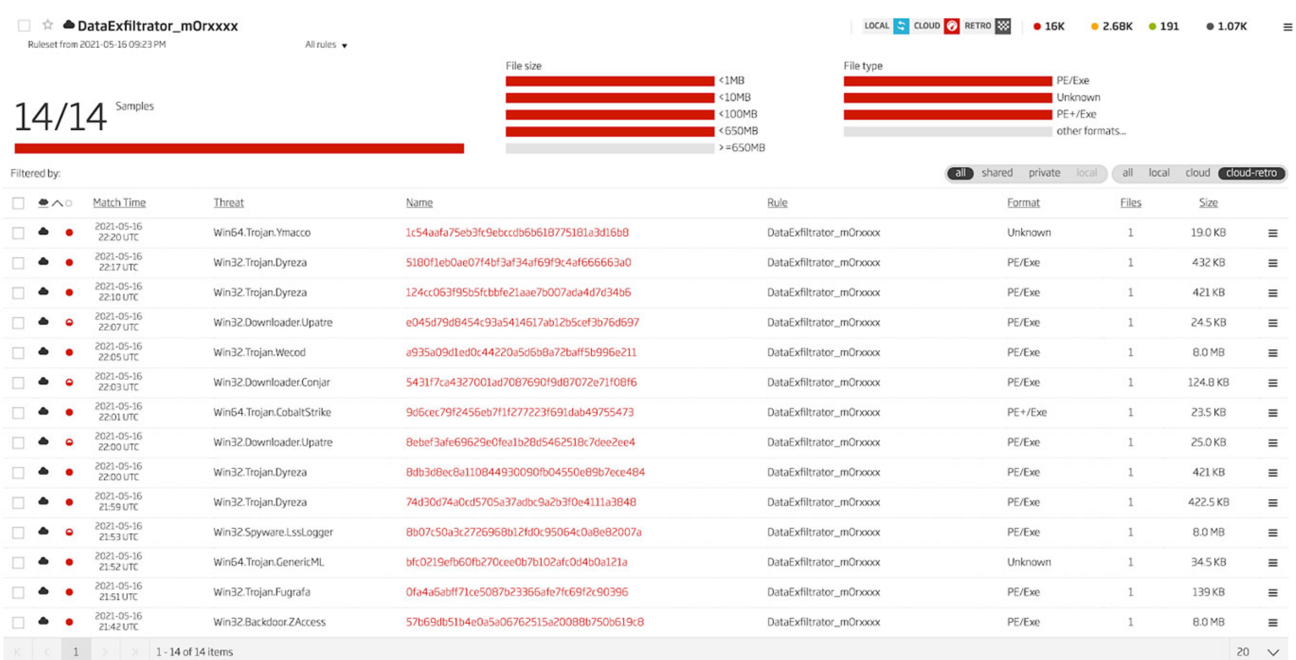

Figure 40: Full String mOrxxxx

The results of retrohunts for this string with two Xs and one X are shown in Figure 41 and 42.

Figure 41: Retrohunt for String with Two Xs

Figure 42: Retrohunt for String with One X

Analysis of the results of these retrohunts will be the topic of a future blog post.

IOCs

Sample 1
File

Filename	Input.exe
Filename	v2c.exe
MD5	1010bec081572dc3bd16e26a1e37d815
SHA1	bfc0219efb60fb270cee0b7b102afc0d4b0a121a
SHA256	dcc4ac1302ac5693875c4a4b193242cbb441b77cd918569c43fe318bcf64fe3d
Import Hash	85ce0801668e488873e72eeb306503da
SSDEEP	768:ycscKP14scGOqEMQcanOPBbEbeFpUGC/YDR5Ws:yV3cGOqEMQcanOJFpUGC/Y9
Timestamp	2021-04-24T17:34:20Z
PDB	E:\work\proj\file_sender\x64\file_sender.pdb
Magic	PE32+ executable (GUI) x86-64, for MS Windows
File Type	Win64 EXE
File Size	35328
First Seen	2021-04-25

User Agent

```
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) discord/0.0.309
Chrome/83.0.4103.122 Electron/9.3.5 Safari/537.36
```

URL

```
hxxps[://]files[.]pablotech[.]info/uploadFile.php
```

Hostname

```
files[.]pablotech[.]info
```

Domain

Name	Registrar	IANA ID
pablotech[.]info	Hosting Concepts B.V. d/b/a Registrar.eu	1647

Sample 2
File

Filename	Input.exe
Filename	sender.exe
MD5	e3300ec2f31f5730970c5bb066d2f0ed
SHA1	c768882e102a5dd3d1c17d306698c5cfc3d9d8d5
SHA256	68af250429833d0b15d44052637caec2afbe18169fee084ee0ef4330661cce9c
Import Hash	6473877da5764bbd5a9b16892ef13b69
SSDEEP	768:zp2FXczP/cpWyB/3RtUcGOqEMIcqfz/YghUx:zp2FsTcB/UcGOqEMIcqfz/Yg4
Timestamp	2021-04-28T03:00:08Z
Magic	PE32+ executable (GUI) x86-64, for MS Windows
File Type	Win64 EXE
File Size	36352
First Seen	2021-05-03

User Agent

Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) discord/0.0.309 Chrome/83.0.4103.122 Electron/9.3.5 Safari/537.36

URL

hxxps[://]figures[.]pablotech[.]info/uploadFile.php

Hostname

figures[.]pablotech[.]info

Domain

Name	Registrar	IANA ID
pablotech[.]info	Hosting Concepts B.V. d/b/a Registrar.eu	1647

Sample 3

File

Filename	Input.exe
Filename	v2c.exe
MD5	4af8b45c9b0f73d47a499d92064b6c2e
SHA1	424f3c281f46e4cf2350c78cfa89a87873e0b994
SHA256	934c557e52bd47fa312ea4098e05781145d0b81c9dc543ef42b266813bdb05d4
Import Hash	6473877da5764bbd5a9b16892ef13b69
SSDEEP	768:9vutX7Qp6CPRp8Yh/ZYWcGOqEMUcgk9/Y7hCeUpU:K7QpJp8YFrcGOqEMUcg0/Y7lk
Timestamp	2021-05-17T20:33:40Z
Magic	PE32+ executable (GUI) x86-64, for MS Windows
File Type	Win64 EXE
File Size	36352
First Seen	2021-05-18

User Agent

Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0

URL

hxxps[://]51.161.82[.]135/uploadFile.php

IP Address

IP	Owner	ASN
51.161.82[.]135	OVH SAS	16276

Sample 4
File

Filename	Input.exe
Filename	v2.exe
MD5	7c801e3c256d2e9e1f4462fe84e44c68
SHA1	4cd9cecd1d093f290e6f8f0ad6d5e76dbedbf3d9
SHA256	a7cf0f72bb6f1e0a61fbf39e3a3a36db6540250caeef35b47fb51a8959f40984
Import Hash	9f86f12427bca134faaa21bcc0d849d3
SSDEEP	768:vkcGOqEMccVhPO4TrASVqipOHMd6m/YFh50:ccGOqEMccV7rAZipOHA/YFT
Timestamp	2021-05-24T23:06:16Z
PDB	E:\work\proj\file_sender\x64\file_sender.pdb
Magic	PE32+ executable (GUI) x86-64, for MS Windows
File Type	Win64 EXE
File Size	37376
First Seen	2021-06-01

User Agent

Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0

URL

hxxps[://]51.161.82[.]135/uploadFile.php

IP Address

IP	Owner	ASN
51.161.82[.]135	OVH SAS	16276

Sample 5
File

Filename	file_sender.exe
Filename	sender.exe
MD5	12a7595d94e142847a04f11659ed183d
SHA1	f80a2f102ca0297d053c75e0676049dc87cb3f35
SHA256	8cfd554a936bd156c4ea29dfd54640d8f870b1ae7738c95ee258408eef0ab9e6
Import Hash	9f86f12427bca134faaa21bcc0d849d3
SSDEEP	768:sPcGOqEMccNNPayYDcfHyIY2QUy2h08wx:2cGOqEMccNEDuhY2FS84
Timestamp	2021-06-15T10:57:36Z
PDB	E:\work\proj\file_sender\x64\file_sender.pdb
Magic	PE32+ executable (GUI) x86-64, for MS Windows
File Type	Win64 EXE
File Size	36352
First Seen	2021-06-16

User Agent

Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0

URL

hxxp[://]51.77.110[.]6/uploadFile.php

IP Address

IP	Owner	ASN
51.77.110[.]6	OVH SAS	16276

YARA Rule

```
private rule WindowsPE
{
  condition:
    uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550
}

rule DataExfiltrator_Decoder
{
  meta:
    author = "Malware Utkonos"
    date = "2021-05-07"
    description = "String decoding function found in data exfiltration malware."
exemplar = "dcc4ac1302ac5693875c4a4b193242cbb441b77cd918569c43fe318bcf64fe3d"
  strings:
    $a = { 4489442418 // mov dword [rsp+0x18 {arg_18}], r8d
        88542410 // mov byte [rsp+0x10 {arg_10}], dl
        48894c2408 // mov qword [rsp+0x8 {arg_8}], rcx
        4883ec28 // sub rsp, 0x28
        8b442440 // mov eax, dword [rsp+0x40 {arg_18}]
        33d2 // xor edx, edx {0x0}
        b904000000 // mov ecx, 0x4
        48f7f1 // div rcx
        }
  condition:
    WindowsPE and $a
}
```

References:

[1] https://research.checkpoint.com/2020/ransomware-evolved-double-extortion/
[2] https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.html#_5ol9xlbbnrdn
[3] https://github.com/MBCProject/mbc-markdown/tree/master/yfaq
[4] https://github.com/MBCProject/mbc-markdown/blob/master/anti-behavioral-analysis/evade-debugger.md
[5] https://x64dbg.com/
[6] https://medium.com/walmartglobaltech/trickbot-crews-new-cobaltstrike-loader-32c72b78e81c
[7] Ibid.
[8] Thanks to Sandor Nemes for assistance in understanding this behavior.
[9] https://www.youtube.com/watch?v=242Tn0IL2jE&t=1053s
[10] https://hexfiend.com/
[11] https://en.cppreference.com/w/c/algorithm/bsearch
[12] https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex
[13] 934c557e52bd47fa312ea4098e05781145d0b81c9dc543ef42b266813bdb05d4
[14] https://en.wikipedia.org/wiki/Cyclic_redundancy_check
[15] https://portswigger.net/burp
[16] https://www.inetsim.org/
[17] dcc4ac1302ac5693875c4a4b193242cbb441b77cd918569c43fe318bcf64fe3d

[18] Ibid.
[19] 68af250429833d0b15d44052637caec2afbe18169fee084ee0ef4330661cce9c
[20] https://www.relyze.com/
[21] 0234f80c6fd3768f9619d6fcd50d775ec686719fcc665007bfd1606bbe787744
[22] https://medium.com/walmartglobaltech/trickbot-crews-new-cobaltstrike-loader-32c72b78e81c
[23] https://twitter.com/ochsenmeier/status/1379546812437118980

MORE BLOG ARTICLES