

Spoofer Saudi Purchase Order Drops GuLoader – Part 2 | FortiGuard Labs

By James Slaughter

Published: 2022-07-12 · Archived: 2026-04-05 22:00:53 UTC

In [part one](#) of this blog, FortiGuard Labs examined a recently discovered e-mail delivered to a coffee company in Ukraine that was seemingly sent by an oil provider in Saudi Arabia. Purporting to contain an attached purchase order, the image of a PDF file was actually a link to an ISO file hosted in the cloud that contained an executable for GuLoader. What makes this case interesting is that this executable uses NSIS ([Nullsoft Scriptable Install System](#)) to deploy itself.

GuLoader (also known as CloudEye and vbdropper) dates to at least 2019 and is generally used to deploy other malware variants such as Agent Tesla, Formbook, and Lokibot.

In this second part of the series, I will showcase a dynamic analysis of the main file, PO#23754-1.exe, as well as investigate the shellcode file “rudesbies.Par”. It will also highlight some of the defences it puts in place to hinder analysis.

Affected Platforms: Windows

Impacted Users: Windows users

Impact: Potential to deploy additional malware for additional purposes

Severity Level: Medium

PO#23754-1.exe dynamic analysis

This sample has a basic level of awareness of its surroundings. If it is executed in a virtual environment that has an obvious artefact (e.g., VirtualBox Guest Additions Tray), it will halt immediately.

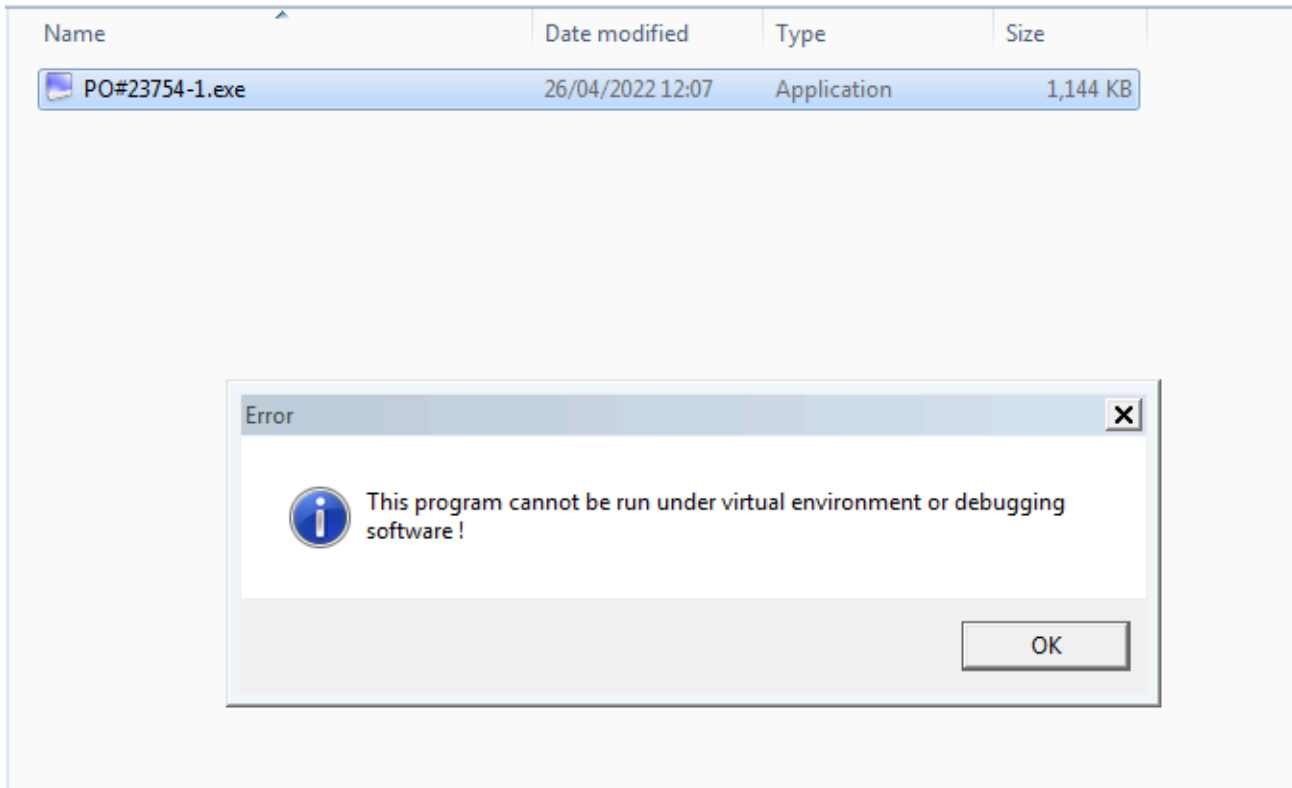


Figure 1. Halting upon detection of a virtual environment.

As the file is executed from inside a mounted ISO file, the execution path will flow from Explorer.

The image shows a screenshot of Windows Task Manager's 'Processes' tab. It displays a list of running processes with columns for Name, CPU, Private, Working Set, Session ID, Name, and Company Name. The process explorer.exe is highlighted in red, and its child process PO#23754-1.exe is highlighted in blue.

Name	CPU	Private	Working Set	Session ID	Name	Company Name
explorer.exe	< 0.01	117,656 K	148,820 K	4508	Windows Explorer	Microsoft Corporation
procexp.exe	4,148	4,148 K	8,632 K	1192	Sysinternals Process Explorer	Sysinternals - www.sysinter...
PO#23754-1.exe	25.01	12,724 K	42,456 K	840	TeraByte Unlimited	Appcelerator, Inc.

Figure 2. Execution tree.

By deploying from within a container such as an ISO, it is often possible to bypass MOTW ([Mark-of-the-Web](#)) controls that would stop files downloaded from the internet from executing.

Since this is a NSIS file, anyone viewing the screen at the time of execution will see also see a progress window as files are installed.

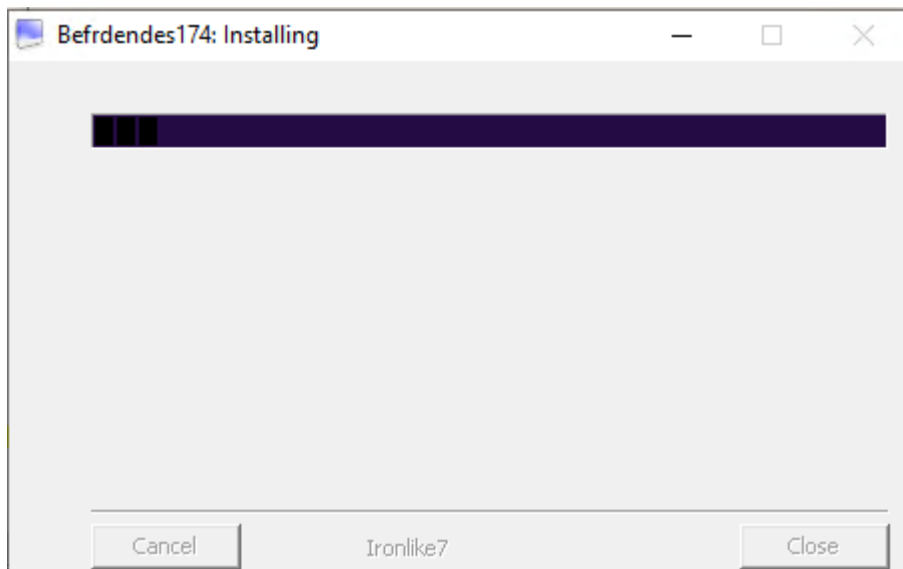


Figure 3. Installation window.

The label use, as shown in the NSIS script from part one of the blog, now becomes apparent when looking at the title bar for the window as well as the lower centre.

```
[NSIS].nsi
1 ; NSIS script (UTF-8) NSIS-3 Unicode
2 ; Install
3
4 Unicode true
5 SetCompressor /SOLID lzma
6 SetCompressorDictSize 8
7
8 ; -----
9 ; HEADER SIZE: 10782
10 ; START HEADER SIZE: 300
11 ; MAX STRING LENGTH: 1024
12 ; STRING CHARS: 1898
13
14 OutFile [NSIS].exe
15 !include WinMessages.nsh
16
17 InstProgressFlags colored
18 InstallDirRegKey HKLM Software\AFSBENDES\Monogene117 UNFATHER
19 InstallColors 000000 250B43
20
21 ; -----
22 ; LANG TABLES: 1
23 ; LANG STRINGS: 48
24
25 BrandingText Ironlike7
26
27 ; LANG: 1033
28 LangString LSTR_0 1033 Ironlike7
29 LangString LSTR_1 1033 Befrdendes174
30 LangString LSTR_5 1033 "Can't write: "
31 LangString LSTR_8 1033 "Could not find symbol: "
32 LangString LSTR_9 1033 "Could not load: "
33 LangString LSTR_13 1033 "Delete file: "
34 LangString LSTR_14 1033 "Delete on reboot: "
35 LangString LSTR_17 1033 "Error decompressing data! Corrupted installer?"
36 LangString LSTR_21 1033 "Extract: "
37 LangString LSTR_22 1033 "Extract: error writing to file "
38 LangString LSTR_24 1033 "No OLE for: "
39 LangString LSTR_25 1033 "Output folder: "
40 LangString LSTR_26 1033 "Remove folder: "
```

Figure 4. Script image from part one showing the title bar and label use.

As was suspected during static analysis, all identified files have been placed into the “Temp” directory of the active user account.

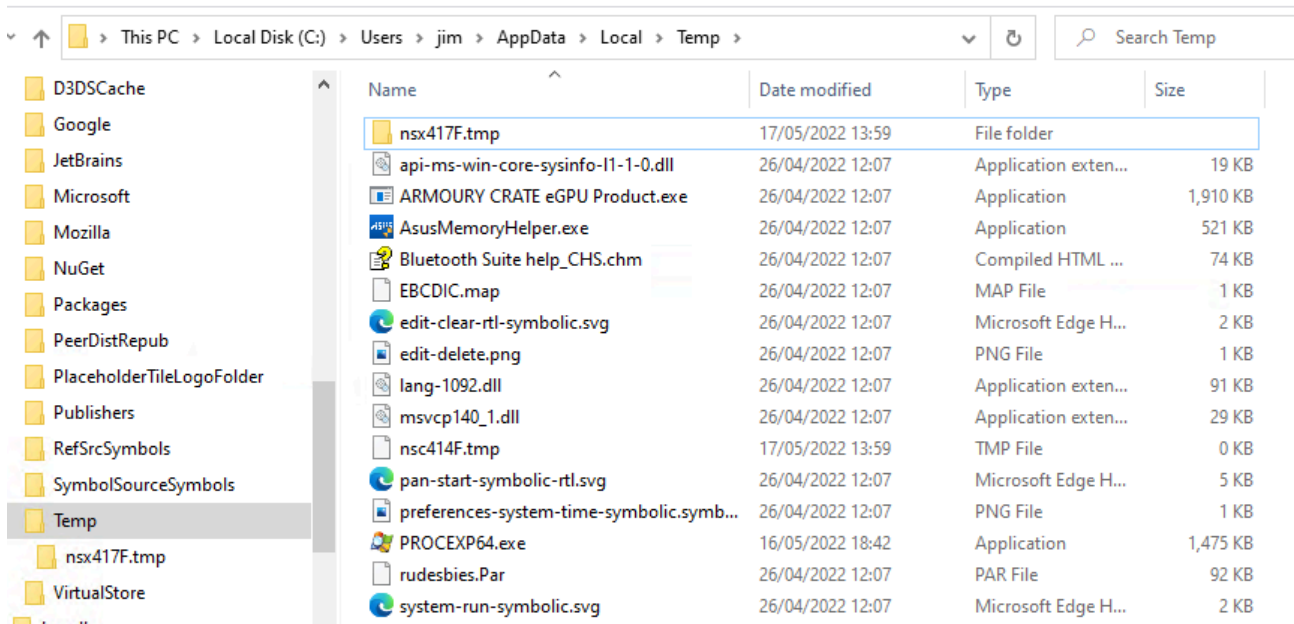


Figure 5. Files installed in \$TEMP (note, PROCEXP.exe is not deployed by this malware).

A unique directory name is created (always beginning with “ns”) to store “System.dll”. A temp file that also is uniquely named (and again always prefixed with “ns”) is used in the NSIS deployment process.

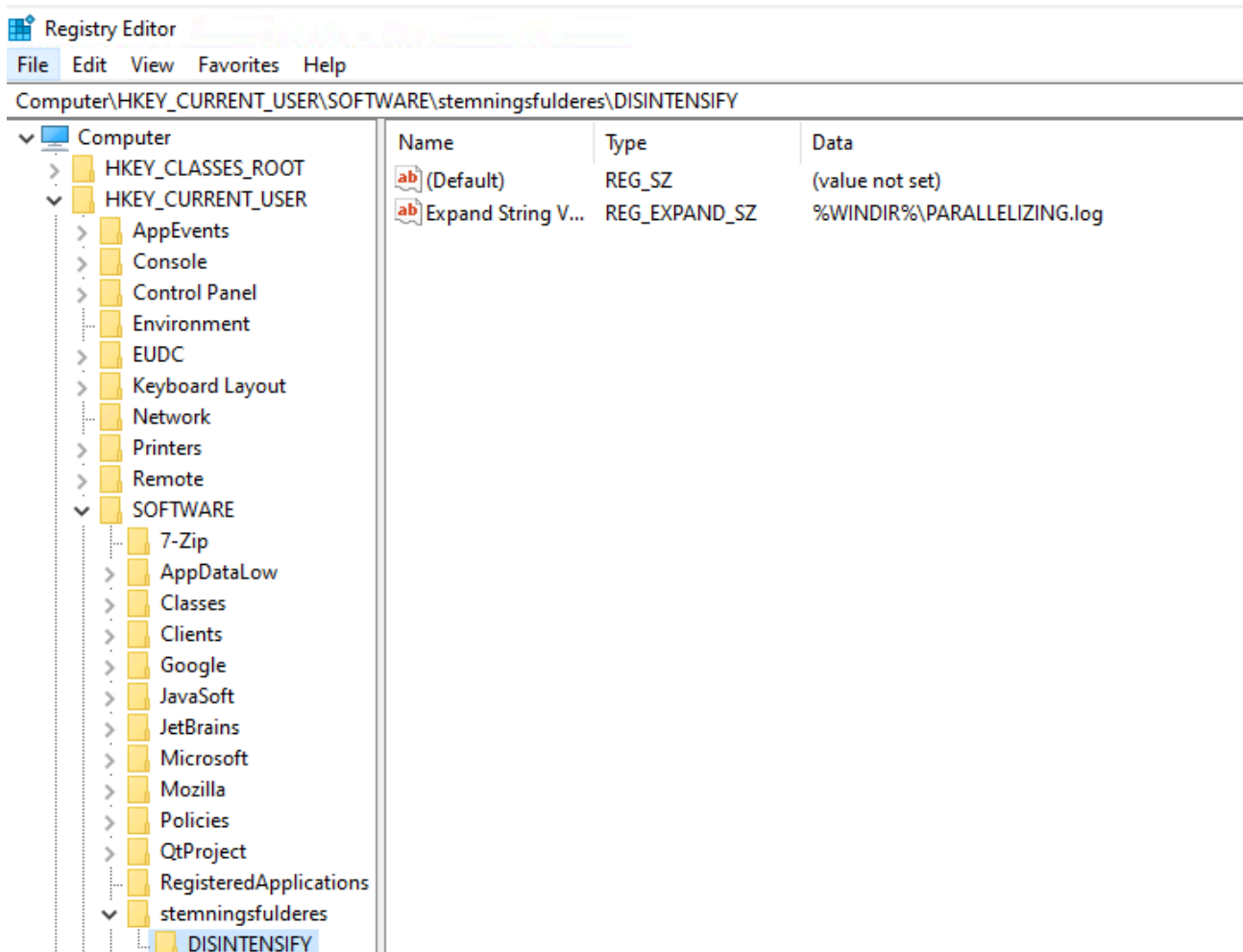


Figure 6. Registry entry created in HKCU\SOFTWARE.

In part one of the blog, a registry key was highlighted in the NSIS script. Figure 6 shows this when implemented on a victim system. It does not appear, however, that this is referenced later by the malware, nor does it appear that a file named “PARALLELIZING.log” is ever committed to disk. It is possible that this is a remnant of the testing phase of the NSIS file that was no longer needed for the active campaign.

System.dll

As mentioned earlier, “System.dll” is dropped into a unique directory. On its own, “System.dll” is a non-malicious file. However, the NSIS script will effectively use this DLL to proxy Windows API calls to “kernel32.dll”. This is done to read the file “rudesbies.Par” and inject its contents into the already running NSIS process. Doing things in this fashion makes it more difficult to trace where the origin of the call is coming from, thereby making it appear to be legitimate.

```

.text:10001817
.text:10001817 ; ===== SUBROUTINE =====
.text:10001817
.text:10001817 ; Attributes: bp-based frame
.text:10001817
.text:10001817      public Call
.text:10001817      call      proc near          ; DATA XREF: .rdata:off_100042D8.i
.text:10001817
.text:10001817      String1      = word ptr -84h
.text:10001817      var_20        = byte ptr -20h
.text:10001817      arg_4         = dword ptr  0Ch
.text:10001817      arg_8         = dword ptr  10h
.text:10001817      arg_C         = dword ptr  14h
.text:10001817      arg_10        = dword ptr  18h
.text:10001817
.text:10001817      push      ebp

```

Figure 7. Call function as seen in IDA.

As can be seen in Figure 7, the “Call” function is used to actually make the API call to “kernel32.dll”

EIP	EAX	EDI	73CC1817	55	push ebp	call
			73CC1818	8BEC	mov ebp, esp	

Figure 8. Call function as seen in the debugger.

Despite a different offset, i.e., the last four bytes of the address in Figure 7 (1817), it can be seen in Figure 8 when “Call” is called by the code executing in the debugger.

In all, five calls are made through “System.dll” to “kernel32.dll”. These read “rudesbies.Par” and then allocate appropriate space in memory within the running executable. The series of five figures shown below demonstrate the entire journey to make this happen. Note the spelling of kernel as “KERNEL”. Referring again back to part one, this is identical to the representation in the NSIS script.

EIP	EAX	EDI	73CD1E74	FF75 CC	push dword ptr ss:[ebp+34]	[ebp+34]:L"KERNEL32::CreateFile(L" C:\Users\SHEFFI-1\AppData\Local\Temp\rudesbies.Par"
			73CD1E75	8B3D Q440CD73	mov edi, dword ptr ds:[&&10ba1Freeb]	
			73CD1E76	FFD7	call edi	
			73CD1E77	FF75 EC	push dword ptr ss:[ebp-14]	[ebp-14]:L"C:\Users\SHEFFI-1\AppData\Local\Temp\rudesbies.Par"
			73CD1E78	FFD7	call edi	
			73CD1E79	FF75 D4	push dword ptr ss:[ebp-2C]	[ebp-2C]:L"KERNEL32"
			73CD1E7A	FFD7	call edi	

Figure 9. CreateFileW is called to create a handle (a reference in memory within the OS) to the file “rudesbies.Par”.

```

EIP → 73CD1E68 FF75 CC push dword ptr ss:[ebp-34] [ebp-34]:L"KERN132::GetFileSize(1 R1, #1 0)1.r7"
      73CD1E6E 8B3D 0440CD73 mov edi,dword ptr ds:[<&GlobalFreee]
      73CD1E74 FFD7 call edi
      73CD1E76 FF75 EC push dword ptr ss:[ebp-14] [ebp-14]:L"GetFileSize"
      73CD1E79 FFD7 call edi
      73CD1E7B FF75 D4 push dword ptr ss:[ebp-2C] [ebp-2C]:L"KERN132"
      73CD1E7E FFD7 call edi
    
```

Figure 10. GetFileSize is called to determine the amount of memory required to store “rudesbies.Par”when opened.

```

EIP → 73CD1E68 FF75 CC push dword ptr ss:[ebp-34] [ebp-34]:L"KERN132::VirtualAllocEx(-1,1 0,1 0x100000, 1 0x3000, 1 64)p.R3"
      73CD1E6E 8B3D 0440CD73 mov edi,dword ptr ds:[<&GlobalFreee]
      73CD1E74 FFD7 call edi
      73CD1E76 FF75 EC push dword ptr ss:[ebp-14] [ebp-14]:L"VirtualAllocEx"
      73CD1E79 FFD7 call edi
      73CD1E7B FF75 D4 push dword ptr ss:[ebp-2C] [ebp-2C]:L"KERN132"
      73CD1E7E FFD7 call edi
    
```

Figure 11. VirtualAllocEx is called to prepare a designated area of memory to store the contents of “rudesbies.Par”.

```

EIP → 73CD1E68 FF75 CC push dword ptr ss:[ebp-34] [ebp-34]:L"KERN132::ReadFile(1 R1, 1 R3, 1 R7,#1 0, 1 0)"
      73CD1E6E 8B3D 0440CD73 mov edi,dword ptr ds:[<&GlobalFreee]
      73CD1E74 FFD7 call edi
      73CD1E76 FF75 EC push dword ptr ss:[ebp-14] [ebp-14]:L"ReadFile"
      73CD1E79 FFD7 call edi
      73CD1E7B FF75 D4 push dword ptr ss:[ebp-2C] [ebp-2C]:L"KERN132"
      73CD1E7E FFD7 call edi
    
```

Figure 12. ReadFile is called to take the contents of “rudesbies.Par” and store them in memory.

```

EIP → 73CD1E68 FF75 CC push dword ptr ss:[ebp-34] [ebp-34]:L"KERN132::CloseHandle(1 R1)"
      73CD1E6E 8B3D 0440CD73 mov edi,dword ptr ds:[<&GlobalFreee]
      73CD1E74 FFD7 call edi
      73CD1E76 FF75 EC push dword ptr ss:[ebp-14] [ebp-14]:L"CloseHandle"
      73CD1E79 FFD7 call edi
      73CD1E7B FF75 D4 push dword ptr ss:[ebp-2C] [ebp-2C]:L"KERN132"
      73CD1E7E FFD7 call edi
    
```

Figure 13. CloseHandle asks the OS to release the handle to “rudesbies.Par”.

Injection

Once rudesbies.Par is successfully read, it must then be injected back into the running NSIS process. This done into a random region of memory within the NSIS process.

Address	Size	Info	Content	Type	Protection	Initial
03A40000	000FD000	Reserved		PRV		-RW--
03B3D000	00003000	Thread 5A4 Stack		PRV	-RW-G	-RW--
03B40000	00100000			PRV	ERW--	ERW--

Figure 14. By observing the memory map in the debugger, the addition of a region with the above characteristics can be seen.

Viewing that region of memory initially shows what looks to be a very large number of “cmp” operations.

EDX EBX	Address	Hex	Assembly	Comment
	03B40000	80F9 FB	cmp cl,FB	
	03B40003	80F9 6E	cmp cl,6E	6E: 'n'
	03B40006	80F9 CC	cmp cl,CC	
	03B40009	80FB 81	cmp b1,81	
	03B4000C	80FA E5	cmp d1,E5	
	03B4000F	80FB 39	cmp b1,39	39: '9'
	03B40012	80F9 88	cmp cl,88	
	03B40015	80F9 FD	cmp cl,FD	
	03B40018	80FA D5	cmp d1,D5	
	03B4001B	80FB 32	cmp b1,32	32: '2'
	03B4001E	3C EC	cmp al,EC	
	03B40020	3C AC	cmp al,AC	
	03B40022	80F9 3C	cmp cl,3C	3C: '<'
	03B40025	E9 1F010000	jmp 3B40149	
	03B4002A	80FB 7C	cmp b1,7C	7C: ' '
	03B4002D	80FB CE	cmp b1,CE	
	03B40030	80F9 DE	cmp cl,DE	
	03B40033	80F9 5A	cmp cl,5A	5A: 'Z'
	03B40036	80F9 9D	cmp cl,9D	
	03B40039	80FB 11	cmp b1,11	
	03B4003C	80FA 5C	cmp d1,5C	5C: '\\'
	03B4003F	80FB BA	cmp b1,BA	
	03B40042	80F9 B5	cmp cl,B5	
	03B40045	80F9 A1	cmp cl,A1	
	03B40048	5F	pop edi	
	03B40049	3C 55	cmp al,55	55: 'U'
	03B4004B	80F9 28	cmp cl,28	28: '('
	03B4004E	80FB 73	cmp b1,73	73: 's'
	03B40051	80FB EC	cmp b1,EC	
	03B40054	3C B1	cmp al,B1	
	03B40056	80FB 10	cmp b1,10	
	03B40059	80FB 50	cmp b1,50	50: 'P'
	03B4005C	80F9 9F	cmp cl,9F	
	03B4005F	80F9 C9	cmp cl,C9	
	03B40062	80F9 41	cmp cl,41	41: 'A'
	03B40065	31D2	xor edx,edx	
	03B40067	3C 0B	cmp al,0B	0B: 'v'
	03B40069	80FB 4C	cmp b1,4C	4C: 'L'
	03B4006C	80F9 EF	cmp cl,EF	
	03B4006F	80F9 E9	cmp cl,E9	
	03B40072	3C 6D	cmp al,6D	6D: 'm'
	03B40074	3C BD	cmp al,BD	
	03B40076	80F9 42	cmp cl,42	42: 'B'
	03B40079	80FB E7	cmp b1,E7	

Figure 15. Initial view of the injected shellcode.

Shellcode

The instructions shown in Figure 15 are junk code and don't actually do anything. Regardless of the memory offset used to load the shellcode, the first set of meaningful instructions start at 014E.

EIP	Address	Hex	Assembly	Comment
	00020149	E8 DCFEFFFF	call 2002A	
	0002014E	C7	???	
	0002014F	209F 912ED674	and byte ptr ds:[edi+74D62E91],b1	
	00020155	06	push es	
	00020156	5A	pop edx	
	00020157	D4 02	aam 2	
	00020159	70 3A	jo 20195	
	00020158	A1 23EF15DF	mov eax,dword ptr ds:[DF15EF23]	
	00020160	B1 FC	mov cl,FC	
	00020162	9D	popfd	
	00020163	46	inc esi	
	00020164	30F9	xor cl,bh	
	00020166	52	push edx	
	00020167	E2 5C	loop 201C5	
	00020169	24 B9	and al,B9	
	00020168	EF	out dx,eax	
	0002016C	7C C3	jl 20131	
	0002016E	90	nop	
	0002016F	29CA	sub edx,ecx	
	00020171	D05494 4D	rc1 byte ptr ss:[esp+edx*4+4D],1	
	00020175	AC	lodsb	
	00020176	6A 6C	push 6C	
	00020178	3D 1834795A	cmp eax,5A793418	
	0002017D	B6 AB	mov dh,AB	
	0002017F	77 AB	ja 2012C	
	00020181	40	inc eax	
	00020182	D0E0	shl al,1	
	00020184	66:80E9 BA	sub cl,BA	
	00020188	56	push esi	
	00020189	CC	int3	
	0002018A	B3 84	mov b1,84	
	0002018C	77 6A	ja 201F8	

Figure 16. Start of actual instruction set (0002014E in this example, still in its encoded format).

through instructions manually versus when running normally, this will create a break in the normal flow of the program.

00020475	^ E1 8B	loope 20402
00020477	84D1	test cl,dl
00020479	0888 6E380B61	or ecx,dword ptr ds:[ebx+610B386E]
0002047F	843B	test byte ptr ds:[ebx],bh
00020481	^ E1 8B	loope 2040E
00020483	8481 F91F12F0	test byte ptr ds:[ecx-FEDE007],a1
00020489	C681 FA66B428 FA	mov byte ptr ds:[ecx+28B466FA],FA
00020490	45	inc ebp
00020491	84E4	test ah,ah
EIP 00020493	43	inc ebx
00020494	0F31	outsc
00020496	39CB	cmp ebx,ecx
00020498	^ 72 BC	jb 20456
0002049A	38EC	cmp ah,ch
0002049C	84D0	test al,dl
0002049E	29CD	sub ebp,ecx
000204A0	84D0	test al,dl
000204A2	C785 4C010000 000000	mov dword ptr ss:[ebp+14C],0
000204AC	39D9	cmp ecx,ebx
000204AE	66:F7C3 1384	test bx,8413
000204B3	E8 02000000	call 204BA
000204B8	85DB	test ebx,ebx
000204BA	84D3	test bl,dl
000204BC	8F45 44	pop dword ptr ss:[ebp+44]
000204BF	84F4	test ah,dh
000204C1	85C0	test eax,eax
000204C3	66:F7C2 EFFD	test dx,FDEF
000204C8	8845 44	mov eax,dword ptr ss:[ebp+44]
000204CB	39C8	cmp eax,ecx
000204CD	^ EB 2C	jmp 204FB
000204CF	6E	outsc

Figure 19. RDTSC instruction.

FortiGuard Labs was unfortunately unable to retrieve the final payload that was to have been dropped by this GuLoader sample. As mentioned earlier, GuLoader has been known to drop different [types of malware](#), such as Agent Tesla, Formbook, and Lokibot.

Conclusion

Despite being a dropper, this sample shows the advantages of when malware developers increase the number of defences against analysis in their code. It slows down analysts attempting to share the details of a campaign, thereby maximizing the amount of time a malicious actor can keep their infrastructure up before becoming known and then compromised or taken down.

GuLoader can be adapted for delivery in different campaigns with [different malware types](#). It is an interesting and active threat that will likely continue for some time to come.

Fortinet Protections

The GuLoader sample mentioned in this blog is detected by the following (AV) signature:

NSIS/Injector.AOW!tr

Fortinet customers are protected from this malware through FortiGuard’s Antivirus, and CDR (content disarm and reconstruction) services and FortiMail, FortiClient, and FortiEDR solutions.

Due to the ease of disruption, damage to daily operations, potential impact to the reputation of an organization, and the unwanted destruction or release of personally identifiable information (PII), etc., it is important to keep all AV and IPS signatures up to date.

Fortinet also has multiple solutions designed to help train users to understand and detect phishing threats:

The FortiPhish Phishing Simulation Service uses real-world simulations to help organizations test user awareness and vigilance to phishing threats and to train and reinforce proper practices when users encounter targeted phishing attacks.

In addition, we suggest that organizations also have their end users go through our FREE NSE training: NSE 1 – Information Security Awareness. It includes a module on Internet threats that is designed to help end users learn how to identify and protect themselves from various types of phishing attacks.

IOCs

Filename	SHA256
PO#23754-1.ISO	c4debbf9c0ec8a56aea5cd97215c6c906bd475ea8bd521fb9a346a4c992a0448
PO#23754-1.exe	14d52119459ef12be3a2f9a3a6578ee3255580f679b1b54de0990b6ba403b0fe
rudesbies.Par	4a1b6b30209c35ab180fa675a769e3285f54597963dd0bb29f7adb686ba88b79
GuLoader	344362b48b8aa9a89623e0bfd139d62f07e2523e600a79bb5af940f35d0740e5
GuLoader	3e79ce8ac441c8c8e777fe0804b67da0bd908a045d553a31893d95f15ae4ea01
GuLoader	9c5f99c37d042b0d6f2b5614fade06d373b2b954bf021bbf955df03693f2380d
GuLoader	53a0111fa7fca816618b65709ebf5d04ae9a64f9ebcfe08c60117a6a6f9d8030
GuLoader	5805e51dc4825c86b2d38c2a011429259954395e2d7b1fd06d83a2a3ec16fc14
GuLoader	1051d3690e70e4227a2b0a0aa87367fb09c49c55360c7a1880b2acfba0b77490
GuLoader	cc1ad7582d16db389c1b15a1cccdc188a85398165623876f4c7887743e54a9f9

Network IOCs

bounceclick[.]live/VVB/COrg_RYGGqN229.binb

Thanks to Fred Gutierrez who helped contribute to this blog.

Learn more about Fortinet's [FortiGuard Labs](#) threat research and intelligence organization and the FortiGuard Security Subscriptions and Services [portfolio](#).

Source: <https://www.fortinet.com/blog/threat-research/spoofed-saudi-purchase-order-drops-guloader-part-two>