Unloading the GuLoader

labs.vipre.com/unloading-the-guloader/

Posted by **VIPRE Labs**

We recently came across a spike of spam email samples containing GuLoader. This malware was discovered last year in 2019 and became more popular among cyber criminals during the coronavirus outbreak. GuLoader is usually attached to a spam email related to bill payments, wire transfers or COVID malspam (you can see a detailed analysis of the COVID malspam here). GuLoader is written in VB5/6 and compressed in a .rar/.iso file. We can see on the graph below the increase of GuLoader which our customers have received:

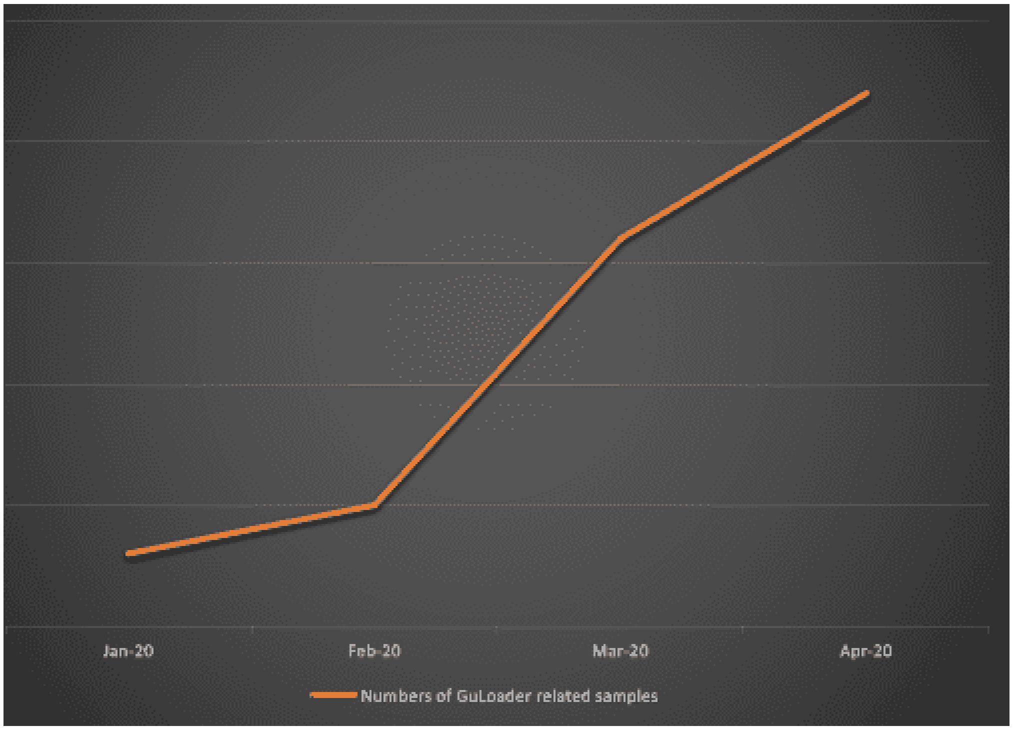
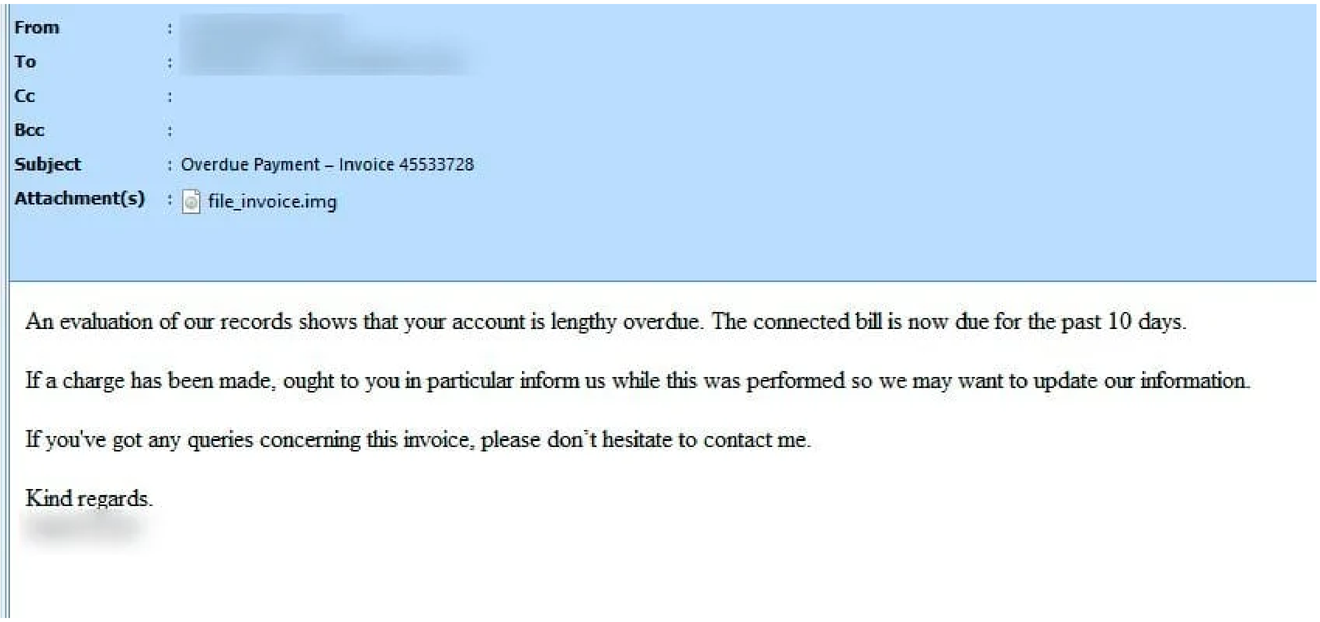


Figure 1.0 Data collected from January to April 2020 showing the increase in GuLoader related samples



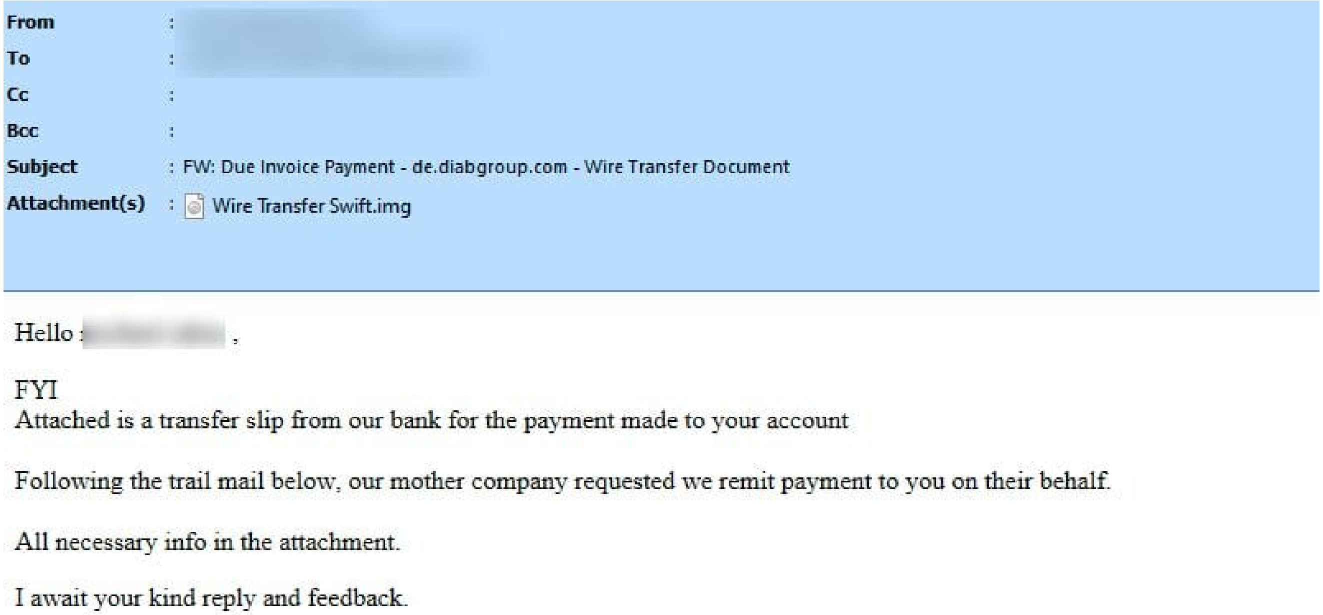
From : [redacted]
To : [redacted]
Cc :
Bcc :
Subject : Overdue Payment – Invoice 45533728
Attachment(s) : file_invoice.img

An evaluation of our records shows that your account is lengthy overdue. The connected bill is now due for the past 10 days.

If a charge has been made, ought to you in particular inform us while this was performed so we may want to update our information.

If you've got any queries concerning this invoice, please don't hesitate to contact me.

Kind regards.
[redacted]



From : [redacted]
To : [redacted]
Cc :
Bcc :
Subject : FW: Due Invoice Payment - de.diabgroup.com - Wire Transfer Document
Attachment(s) : Wire Transfer Swift.img

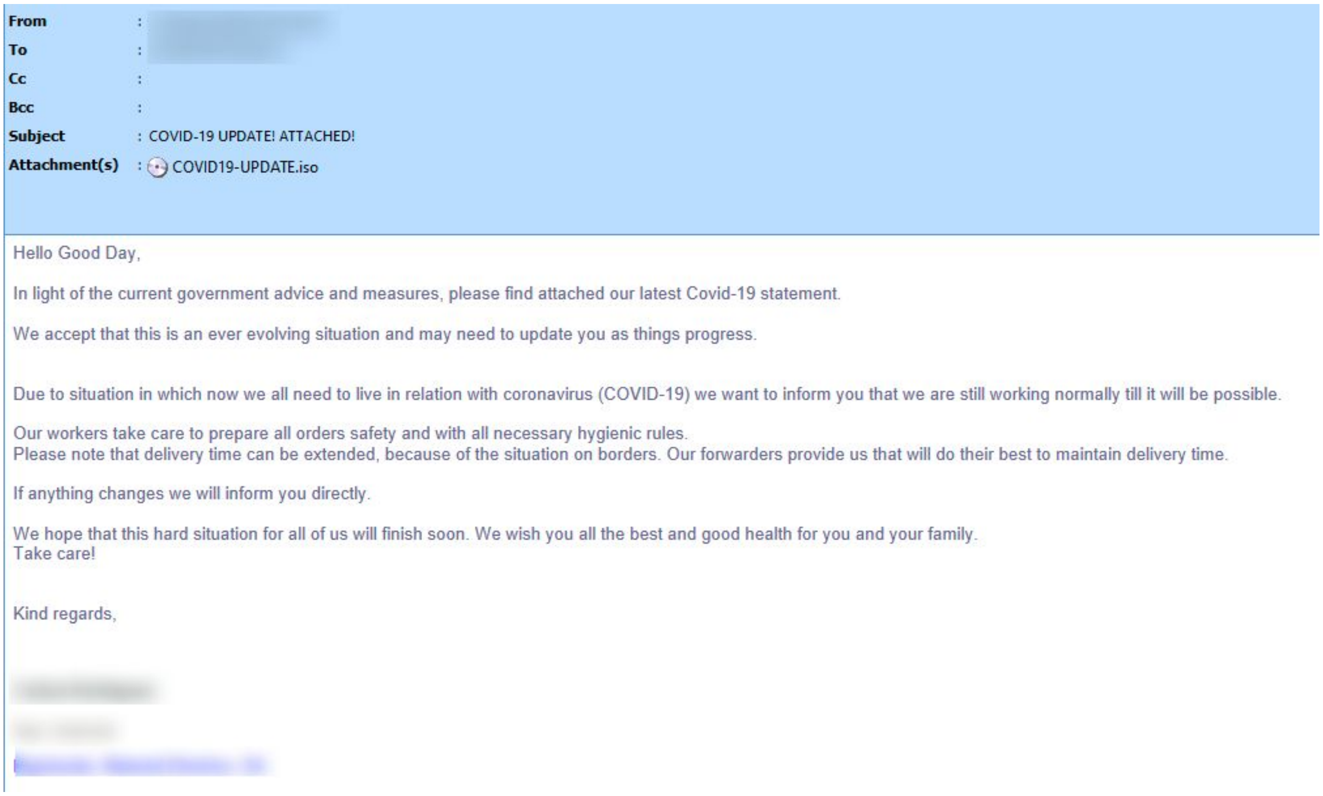
Hello [redacted] ,

FYI
Attached is a transfer slip from our bank for the payment made to your account

Following the trail mail below, our mother company requested we remit payment to you on their behalf.

All necessary info in the attachment.

I await your kind reply and feedback.



From : [redacted]
To : [redacted]
Cc :
Bcc :
Subject : COVID-19 UPDATE! ATTACHED!
Attachment(s) : COVID19-UPDATE.iso

Hello Good Day,

In light of the current government advice and measures, please find attached our latest Covid-19 statement.

We accept that this is an ever evolving situation and may need to update you as things progress.

Due to situation in which now we all need to live in relation with coronavirus (COVID-19) we want to inform you that we are still working normally till it will be possible.

Our workers take care to prepare all orders safety and with all necessary hygienic rules.
Please note that delivery time can be extended, because of the situation on borders. Our forwarders provide us that will do their best to maintain delivery time.

If anything changes we will inform you directly.

We hope that this hard situation for all of us will finish soon. We wish you all the best and good health for you and your family.
Take care!

Kind regards,

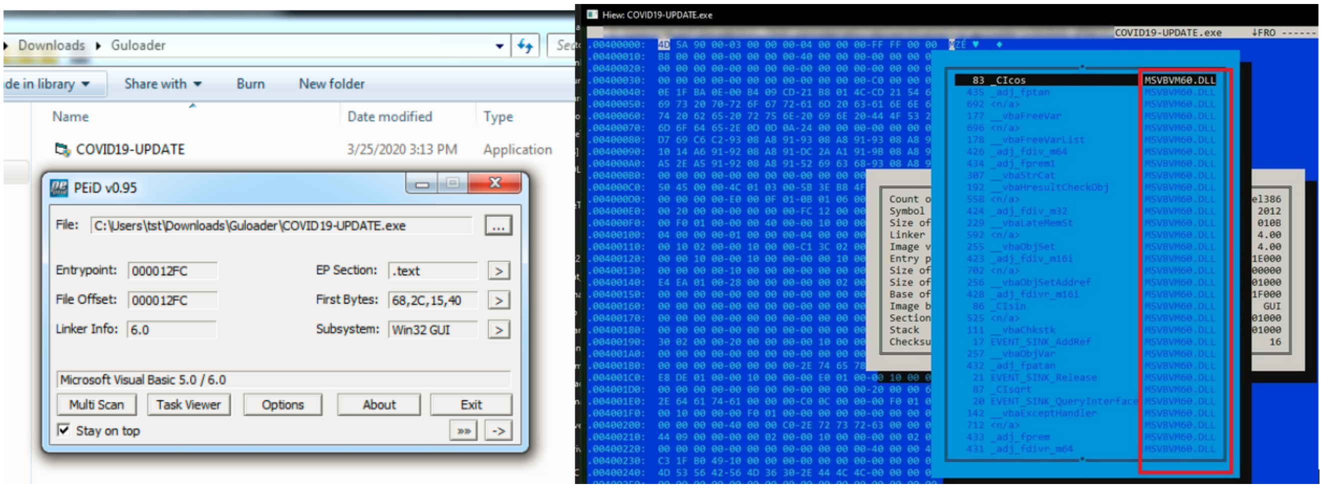
[redacted]

Figure 2.0 Spam emails containing GuLoader

Guloader is popular for distributing Remote Access Trojan (*RAT*) tools. These allow the attackers to control, monitor, or steal information from the infected machine. This malware downloader utilizes cloud hosting services (*Microsoft OneDrive or Google Drive*) to keep its payload encrypted.

Dig Deeper Inside of GuLoader

Analyzing the GuLoader sample, the malware is indeed a VB5/6 executable. Also, a compiled Visual Basic sample can be recognized by an imported DLL called *MSVBVM60.DLL*.



Name	Date modified	Type
COVID19-UPDATE	3/25/2020 3:13 PM	Application

PEiD v0.95

File: C:\Users\tst\Downloads\Guloader\COVID19-UPDATE.exe

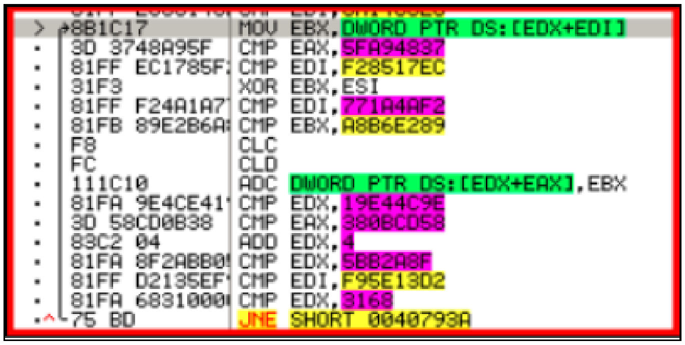
Entrypoint: 000012FC
EP Section: .text
File Offset: 000012FC
First Bytes: 68,2C,15,40
Linker Info: 6.0
Subsystem: Win32 GUI

Microsoft Visual Basic 5.0 / 6.0

```
COVID19-UPDATE.exe
  83 _CIcos                          MSVBVM60.DLL
 435 _adj_fptan                      MSVBVM60.DLL
 692 <n/a>                           MSVBVM60.DLL
 177 __vbaFreeVar                    MSVBVM60.DLL
 696 <n/a>                           MSVBVM60.DLL
 178 __vbaFreeVarList                MSVBVM60.DLL
 426 _adj_fdiv_m64                   MSVBVM60.DLL
 434 _adj_fprem1                     MSVBVM60.DLL
 307 __vbaStrCat                     MSVBVM60.DLL
 192 __vbaHresultCheckObj            MSVBVM60.DLL
 558 <n/a>                           MSVBVM60.DLL
 424 _adj_fdiv_m32                   MSVBVM60.DLL
 229 __vbaLateMemSt                  MSVBVM60.DLL
 592 <n/a>                           MSVBVM60.DLL
 255 __vbaObjSet                     MSVBVM60.DLL
 423 _adj_fdiv_m16i                  MSVBVM60.DLL
 702 <n/a>                           MSVBVM60.DLL
 256 __vbaObjSetAddref               MSVBVM60.DLL
 428 _adj_fdivr_m16i                 MSVBVM60.DLL
  86 _CIsin                          MSVBVM60.DLL
 525 <n/a>                           MSVBVM60.DLL
 111 __vbaChkstk                     MSVBVM60.DLL
  17 EVENT_SINK_AddRef               MSVBVM60.DLL
 257 __vbaObjVar                     MSVBVM60.DLL
 432 _adj_fpatan                     MSVBVM60.DLL
  21 EVENT_SINK_Release              MSVBVM60.DLL
  87 _CIsqrt                         MSVBVM60.DLL
  20 EVENT_SINK_QueryInterface       MSVBVM60.DLL
 142 __vbaExceptHandler              MSVBVM60.DLL
 712 <n/a>                           MSVBVM60.DLL
 433 _adj_fprem                      MSVBVM60.DLL
 431 _adj_fdivr_m64                  MSVBVM60.DLL
```

Figure 3.0 GuLoader sample written in VB5/6 and the msvbvm60.dll

Analyzing further, we've found the malware's encrypted malicious code. This malware allocates virtual memory and decrypts the encrypted malicious code using XOR.



```
> ┌>8B1C17          MOV EBX,DWORD PTR DS:[EDX+EDI]
. │ 3D 3748A95F     CMP EAX,5FA94837
. │ 81FF EC1785F:   CMP EDI,F28517EC
. │ 31F3            XOR EBX,ESI
. │ 81FF F24A1A7'   CMP EDI,771A4AF2
. │ 81FB 89E2B6A:   CMP EBX,A8B6E289
. │ F8              CLC
. │ FC              CLD
. │ 111C10          ADC DWORD PTR DS:[EDX+EAX],EBX
. │ 81FA 9E4CE41'   CMP EDX,19E44C9E
. │ 3D 58CD0B38     CMP EAX,380BCD58
. │ 83C2 04         ADD EDX,4
. │ 81FA 8F2ABB0!   CMP EDX,5BB2A8F
. │ 81FF D2135EF'   CMP EDI,F95E13D2
. │ 81FA 6831000(   CMP EDX,3168
.^└75 BD            JNE SHORT 0040793A
```

Figure 4.0 The decryption routine

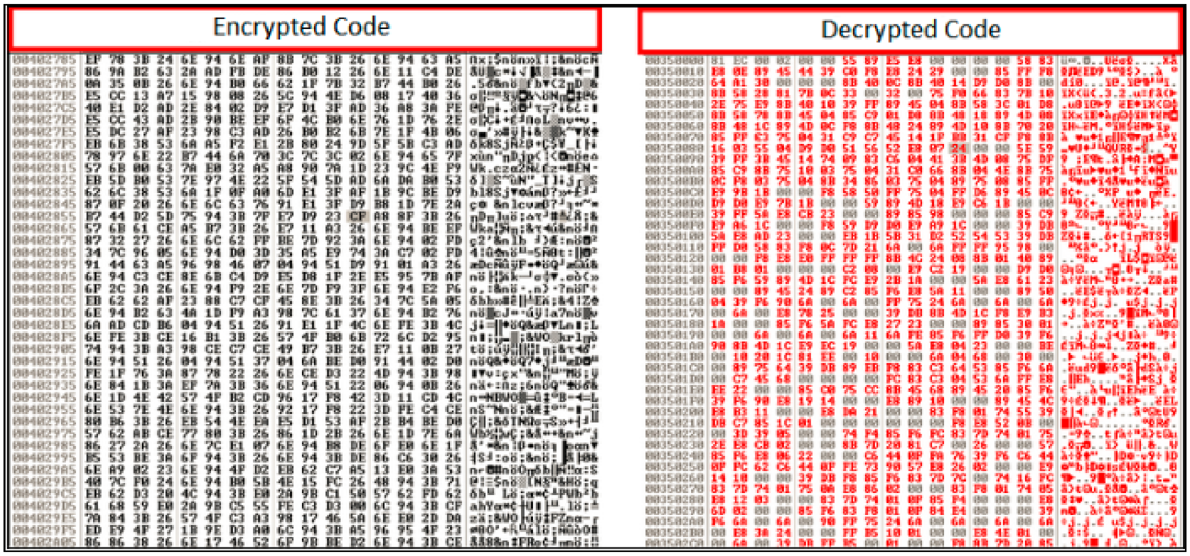
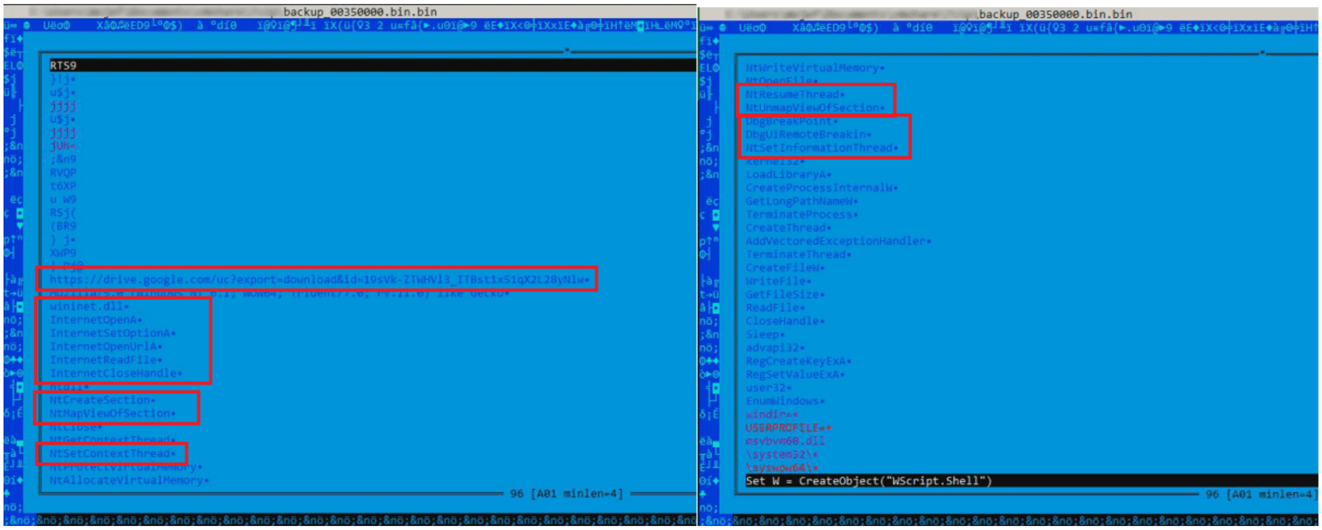


Figure 5.0 The encrypted malicious code (left) and the decrypted malicious code in allocated virtual memory (right)

The decrypted code will be in virtual memory 0x350000. Checking this memory in memory map, it has read, write, and execute (*RWE*) access. We've now dumped the decrypted code to conduct analysis.



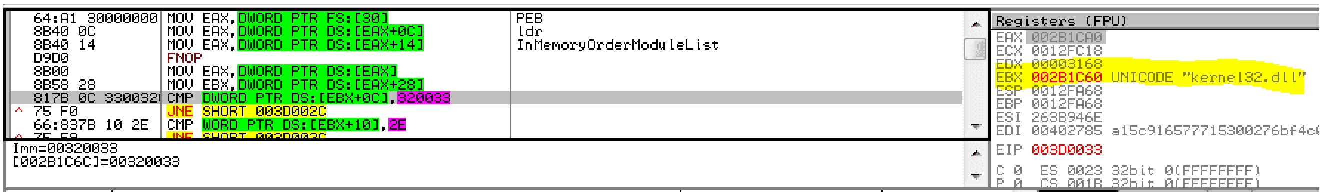
```
backup_00350000.bin.bin
RTS9
}!j•
u$j•
jjjj
u$j•
jjjj
jUh•
;&n9
RVQP
t6XP
u W9
RSj(
(BR9
} j•
XWP9
| Pj@
https://drive.google.com/uc?export=download&id=19sVk-ZTWHVl3_ITBst1x51qX2L28yNlw•
Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko•
wininet.dll•
InternetOpenA•
InternetSetOptionA•
InternetOpenUrlA•
InternetReadFile•
InternetCloseHandle•
ntdll•
NtCreateSection•
NtMapViewOfSection•
NtClose•
NtGetContextThread•
NtSetContextThread•
NtProtectVirtualMemory•
NtAllocateVirtualMemory•
                                                         96 [A01 minlen=4]
```

```
backup_00350000.bin.bin
NtWriteVirtualMemory•
NtOpenFile•
NtResumeThread•
NtUnmapViewOfSection•
DbgBreakPoint•
DbgUiRemoteBreakin•
NtSetInformationThread•
kernel32•
LoadLibraryA•
CreateProcessInternalW•
GetLongPathNameW•
TerminateProcess•
CreateThread•
AddVectoredExceptionHandler•
TerminateThread•
CreateFileW•
WriteFile•
GetFileSize•
ReadFile•
CloseHandle•
Sleep•
advapi32•
RegCreateKeyExA•
RegSetValueExA•
user32•
EnumWindows•
windir=•
USERPROFILE=•
msvbvm60.dll
\system32\•
\syswow64\•
Set W = CreateObject("WScript.Shell")
                                                         96 [A01 minlen=4]
```

Figure 6.0 The dumped memory and the familiar strings that were found in the decrypted code

Checking the strings on the decrypted code, we can see clearly the cloud hosting service URL that stores the encrypted payload (*hxxps://drive[.]google[.]com/uc?export=download&id=19sVk-ZTWHVl3_ITBst1x51qX2L28yNlw*). We can also see familiar DLLs like wininet.dll and APIs like InternetOpenA, InternetOpenUrlA, InternetSetOptionA etc. The wininet.dll contains internet related functions like InternetOpenA and these functions will probably be used to connect to the URL that contains the encrypted payload.

Analyzing what's inside of the decrypted code, we can see that the malware will find the GetProcAddress function in kernel32.dll because GetProcAddress is important in finding and calling other API functions. In order to do this, the malware will first access the Process Environment Block (*PEB*) -> LDR data -> InMemoryOrderModuleList and then get the address of the module kernel32.dll.



```
  64:A1 30000000 MOV EAX,DWORD PTR FS:[30]          PEB
  8B40 0C        MOV EAX,DWORD PTR DS:[EAX+0C]      ldr
  8B40 14        MOV EAX,DWORD PTR DS:[EAX+14]      InMemoryOrderModuleList
  D9D0           FNOP
  8B00           MOV EAX,DWORD PTR DS:[EAX]
  8B58 28        MOV EBX,DWORD PTR DS:[EAX+28]
  817B 0C 330032 CMP DWORD PTR DS:[EBX+0C],320033
^ 75 F0          JNE SHORT 003D002C
  66:837B 10 2E  CMP WORD PTR DS:[EBX+10],2E
^ 75 E9          JNE SHORT 003D002C
Imm=00320033
[002B1C6C]=00320033

Registers (FPU)
EAX 002B1CA0
ECX 0012FC18
EDX 00003168
EBX 002B1C60 UNICODE "kernel32.dll"
ESP 0012FA68
EBP 0012FA68
ESI 263B946E
EDI 00402785 a15c916577715300276bf4c

EIP 003D0033

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
```

Figure 7.0 Accessing the PEB and getting the address of kernel32.dll

After obtaining the address of kernel32.dll and finding GetProcAddress in kernel32.dll, the malware will resolve the following series of APIs:

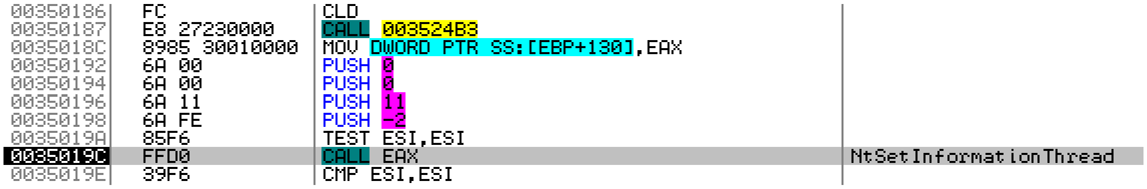
- LoadLibraryA
- TerminateProcess
- EnumWindows
- NtProtectVirtualMemory

- NtSetInformationThread
- NtAllocateVirtualMemory
- DbgBreakPoint
- DbgUiRemoteBreakin

After this, we ran into some anti-analysis techniques. The anti-analysis was used by malware authors to make it more difficult to analyze the malware.

Here are some of the techniques we encountered:

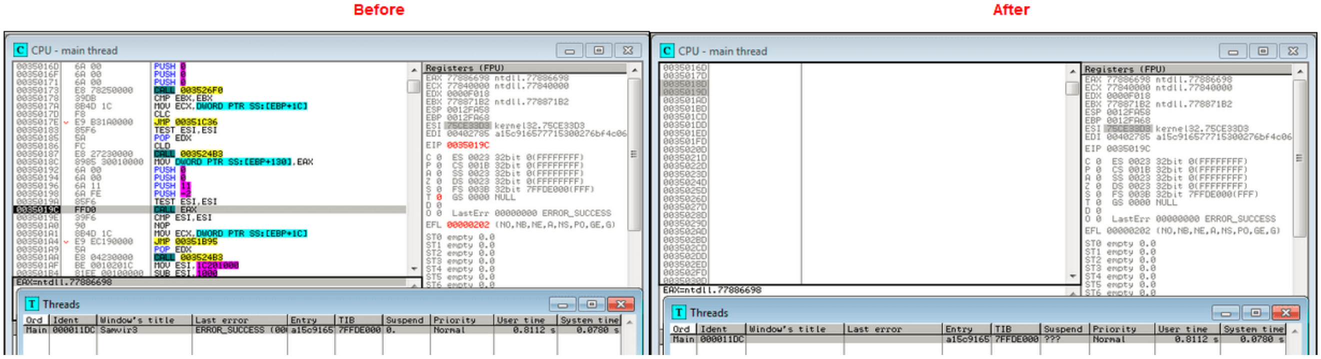
> An anti-debugger that hides the thread from the debugger. In order to perform this, the API NtSetInformationThread is needed. They set the second parameter (*ThreadInformationClass*) to 0x11 which is equivalent to ThreadHideFromDebugger. It will hide the thread from the debugger so it can't be easily debugged. For example, the thread will continue to run, but the debugger will not be able to receive any events related to the thread.



```
00350186  FC             CLD
00350187  E8 27230000    CALL 003524B3
0035018C  8985 30010000  MOV DWORD PTR SS:[EBP+130],EAX
00350192  6A 00          PUSH 0
00350194  6A 00          PUSH 0
00350196  6A 11          PUSH 11
00350198  6A FE          PUSH -2
0035019A  85F6           TEST ESI,ESI
0035019C  FFD0           CALL EAX                               NtSetInformationThread
0035019E  39F6           CMP ESI,ESI
```

Figure 8.0 Calling of NtSetInformationThread to hide the thread from the debugger

> Thread attach in a debugger can be seen in the thread window. On *figure 9.0*, we can see the before and after the thread is hidden from the debugger. The before part is where we can see the main thread and its thread ID which is 11DC. The after part is where the main thread is hidden from the debugger.



Before

CPU - main thread

```
0035016D    6A 00          PUSH 0
0035016F    6A 00          PUSH 0
00350171    6A 00          PUSH 0
00350173    E8 78250000    CALL 003526F0
00350178    39DB           CMP EBX,EBX
0035017A    8B4D 1C        MOV ECX,DWORD PTR SS:[EBP+1C]
0035017D    F8             CLC
0035017E  v E9 B31A0000    JMP 00351C36
00350183    85F6           TEST ESI,ESI
00350185    5A             POP EDX
00350186    FC             CLD
00350187    E8 27230000    CALL 003524B3
0035018C    8985 30010000  MOV DWORD PTR SS:[EBP+130],EAX
00350192    6A 00          PUSH 0
00350194    6A 00          PUSH 0
00350196    6A 11          PUSH 11
00350198    6A FE          PUSH -2
0035019A    85F6           TEST ESI,ESI
0035019C    FFD0           CALL EAX
0035019E    39F6           CMP ESI,ESI
003501A0    90             NOP
003501A1    8B4D 1C        MOV ECX,DWORD PTR SS:[EBP+1C]
003501A4  v E9 EC190000    JMP 00351B95
003501A9    5A             POP EDX
003501AA    E8 04230000    CALL 003524B3
003501AF    BE 0010201C    MOV ESI,1C201000
003501B4    81EE 00100000  SUB ESI,1000
EAX=ntdll.77886698
```

```
Registers (FPU)
EAX 77886698 ntdll.77886698
ECX 77840000 ntdll.77840000
EDX 0000F018
EBX 778871B2 ntdll.778871B2
ESP 0012FA58
EBP 0012FA68
ESI 75CE33D3 kernel32.75CE33D3
EDI 00402785 a15c916577715300276bf4c06

EIP 0035019C

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 00000000 ERROR_SUCCESS

EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
```

Threads

Ord	Ident	Window's title	Last error	Entry	TIB	Suspend	Priority	User time	System time
Main	000011DC	Samvir3	ERROR_SUCCESS (00	a15c9165	7FFDE000	0.	Normal	0.8112 s	0.0780 s

After

CPU - main thread

```
0035016D
0035017D
0035018D
0035019D
003501AD
003501BD
003501CD
003501DD
003501ED
003501FD
0035020D
0035021D
0035022D
0035023D
0035024D
0035025D
0035026D
0035027D
0035028D
0035029D
003502AD
003502BD
003502CD
003502DD
003502ED
003502FD
0035030D
EAX=ntdll.77886698
```

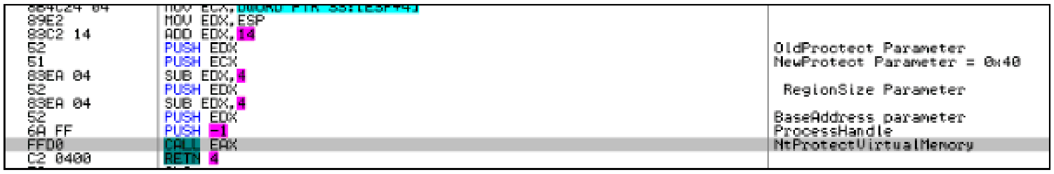
```
Registers (FPU)
EAX 77886698 ntdll.77886698
ECX 77840000 ntdll.77840000
EDX 0000F018
EBX 778871B2 ntdll.778871B2
ESP 0012FA58
EBP 0012FA68
ESI 75CE33D3 kernel32.75CE33D3
EDI 00402785 a15c916577715300276bf4c06

EIP 0035019C

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 00000000 ERROR_SUCCESS

EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
```

Threads

Ord	Ident	Window's title	Last error	Entry	TIB	Suspend	Priority	User time	System time
Main	000011DC			a15c9165	7FFDE000	???	Normal	0.8112 s	0.0780 s

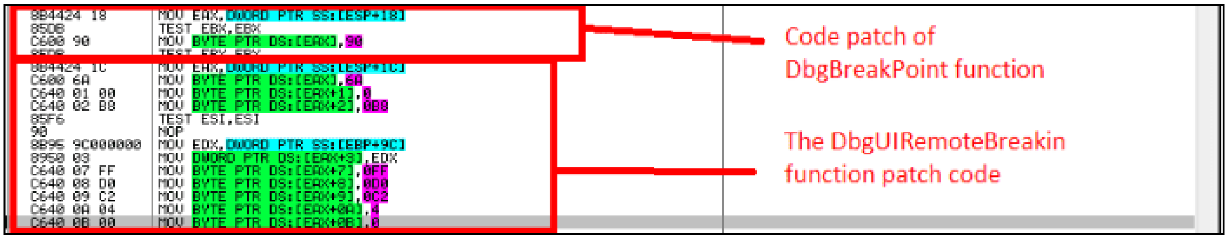
Figure 9.0 Before and after the hiding of thread

There’s another technique that will first call the *NtProtectVirtualMemory* function to set the permission of ntdll’s .text section as *PAGE_EXECUTE_READWRITE*. The ntdll.dll contains the following APIs, *DbgBreakPoint* and *DbgUiRemoteBreakin*, that will be used to perform anti-attach. The malware prevents the debugger from attaching to a process by hooking the *DbgBreakPoint* and *DbgUiRemoteBreakin* functions. For example, it will patch *DbgBreakPoint* and *DbgUIRemoteBreakin* functions that will trigger the process to exit or to designate an unknown location. Like in *figure 12.0*, *DbgUIRemoteBreakin* will call *0x00000000* address and exit.



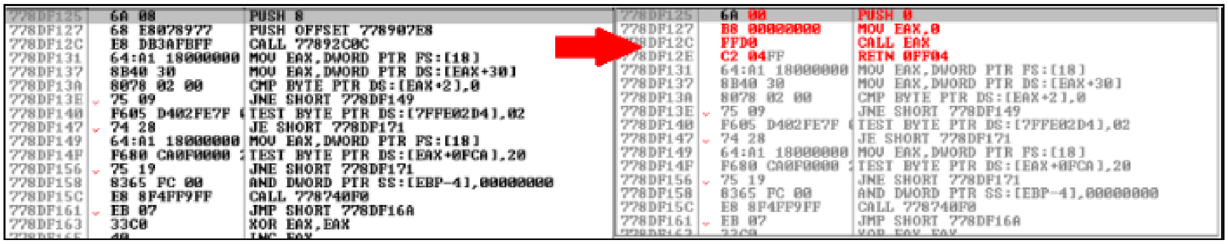
```
8B4C24 04      MOV ECX,DWORD PTR SS:[ESP+4]
89E2           MOV EDX,ESP
83C2 14        ADD EDX,14
52             PUSH EDX                               OldProctect Parameter
51             PUSH ECX                               NewProtect Parameter = 0x40
83EA 04        SUB EDX,4
52             PUSH EDX                                RegionSize Parameter
83EA 04        SUB EDX,4
52             PUSH EDX                               BaseAddress parameter
6A FF          PUSH -1                                ProcessHandle
FFD0           CALL EAX                               NtProtectVirtualMemory
C2 0400        RETN 4
```

Figure 10.0 Calling of NtProtectVirtualMemory to set the permission of NTDLL.DLL



```
8B4424 18      MOV EAX,DWORD PTR SS:[ESP+18]
85DB           TEST EBX,EBX
C600 90        MOV BYTE PTR DS:[EAX],90
85DB           TEST EBX,EBX
8B4424 1C      MOV EAX,DWORD PTR SS:[ESP+1C]
C600 6A        MOV BYTE PTR DS:[EAX],6A
C640 01 00     MOV BYTE PTR DS:[EAX+1],0
C640 02 B8     MOV BYTE PTR DS:[EAX+2],0B8
85F6           TEST ESI,ESI
90             NOP
8B95 9C000000  MOV EDX,DWORD PTR SS:[EBP+9C]
8950 03        MOV DWORD PTR DS:[EAX+3],EDX
C640 07 FF     MOV BYTE PTR DS:[EAX+7],0FF
C640 08 D0     MOV BYTE PTR DS:[EAX+8],0D0
C640 09 C2     MOV BYTE PTR DS:[EAX+9],0C2
C640 0A 04     MOV BYTE PTR DS:[EAX+0A],4
C640 0B 00     MOV BYTE PTR DS:[EAX+0B],0
```

Figure 11.0 Patching of DbgBreakPoint and DbgUiRemoteBreakin for anti-attach technique

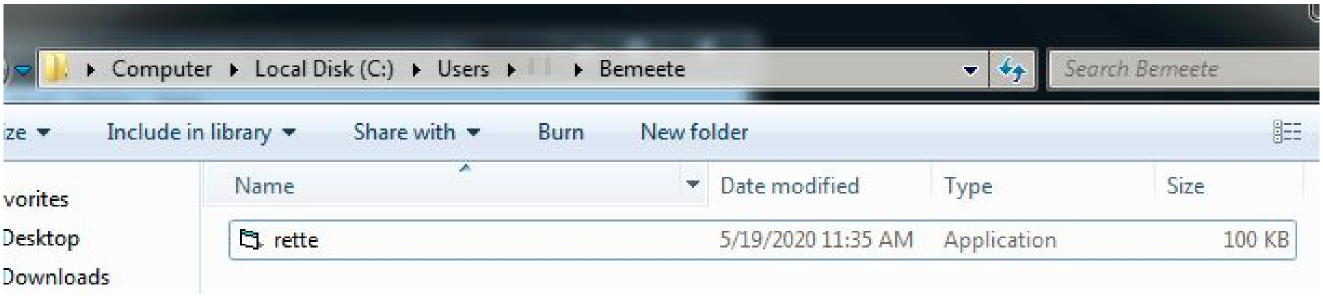


```
778DF125    6A 00            PUSH 0
778DF127    68 E8078977      PUSH OFFSET 778907E8
778DF12C    E8 DB3AFBFF      CALL 77892C0C
778DF131    64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
778DF137    8B40 30          MOV EAX,DWORD PTR DS:[EAX+30]
778DF13A    8078 02 00       CMP BYTE PTR DS:[EAX+2],0
778DF13E    75 09            JNE SHORT 778DF149
778DF140    F605 D402FE7F 0  TEST BYTE PTR DS:[7FFE02D4],02
778DF147    74 28            JE SHORT 778DF171
778DF149    64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
778DF14F    F680 CA0F0000 2  TEST BYTE PTR DS:[EAX+0FCA],20
778DF156    75 19            JNE SHORT 778DF171
778DF158    8365 FC 00       AND DWORD PTR SS:[EBP-4],00000000
778DF15C    E8 8F4FF9FF      CALL 778740F0
778DF161    EB 07            JMP SHORT 778DF16A
778DF163    33C0             XOR EAX,EAX
778DF165    40               INC EAX
```

```
778DF125    6A 00            PUSH 0
778DF127    B8 00000000      MOV EAX,0
778DF12C    FFD0             CALL EAX
778DF12E    C2 04FF          RETN 0FF04
778DF131    64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
778DF137    8B40 30          MOV EAX,DWORD PTR DS:[EAX+30]
778DF13A    8078 02 00       CMP BYTE PTR DS:[EAX+2],0
778DF13E    75 09            JNE SHORT 778DF149
778DF140    F605 D402FE7F 0  TEST BYTE PTR DS:[7FFE02D4],02
778DF147    74 28            JE SHORT 778DF171
778DF149    64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
778DF14F    F680 CA0F0000 2  TEST BYTE PTR DS:[EAX+0FCA],20
778DF156    75 19            JNE SHORT 778DF171
778DF158    8365 FC 00       AND DWORD PTR SS:[EBP-4],00000000
778DF15C    E8 8F4FF9FF      CALL 778740F0
778DF161    EB 07            JMP SHORT 778DF16A
778DF163    33C0             XOR EAX,EAX
```

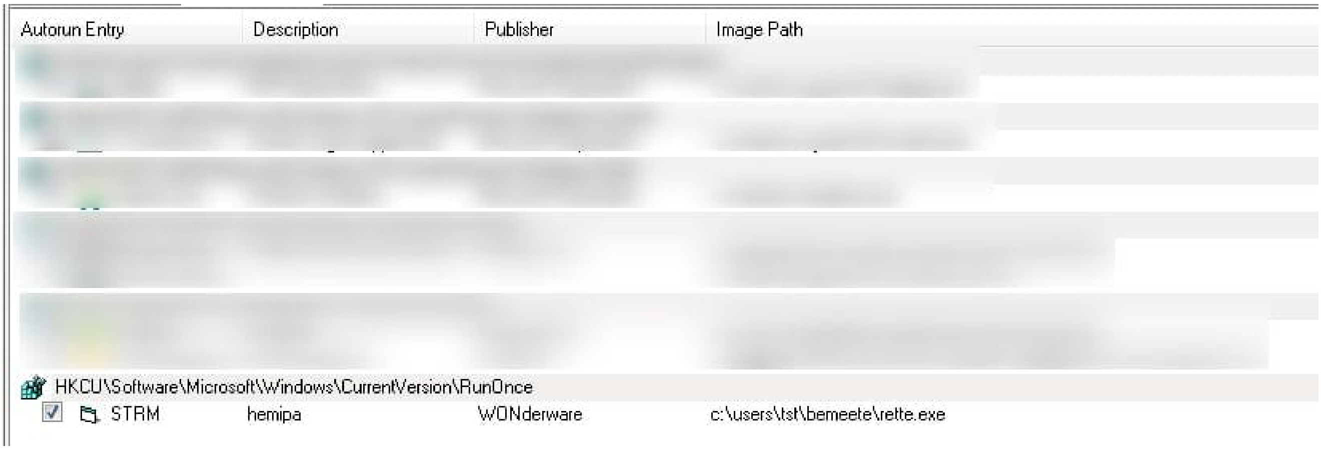
Figure 12.0 *Before and after patching the DbgUIRemoteBreakin function*

GuLoader will create a folder in the C:\Users directory and the created folder contains a copy of the malware itself. It will also achieve persistence by modifying the registry key HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce



Name	Date modified	Type	Size
rette	5/19/2020 11:35 AM	Application	100 KB

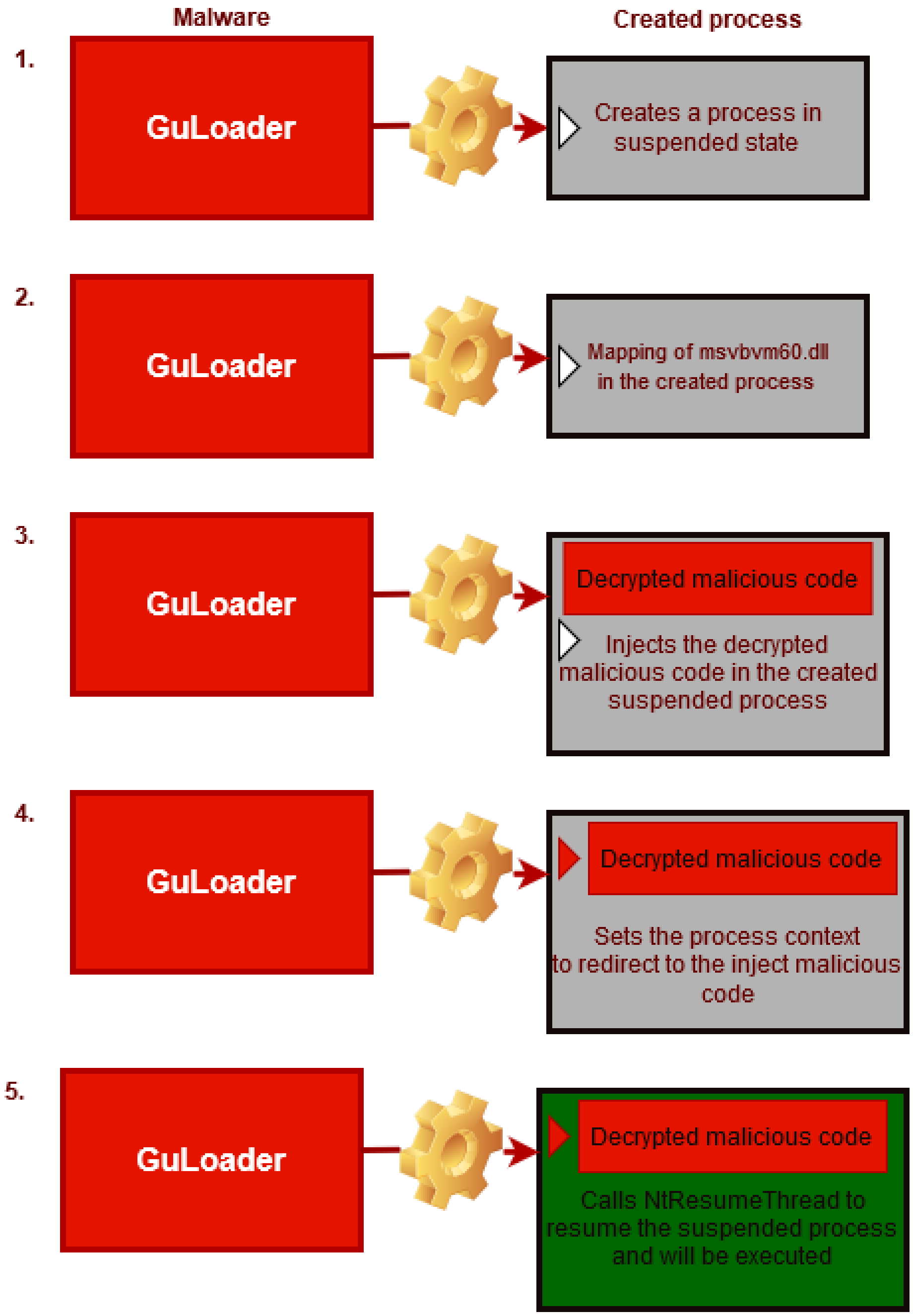
Figure 13.0 The created folder containing the created malware copy



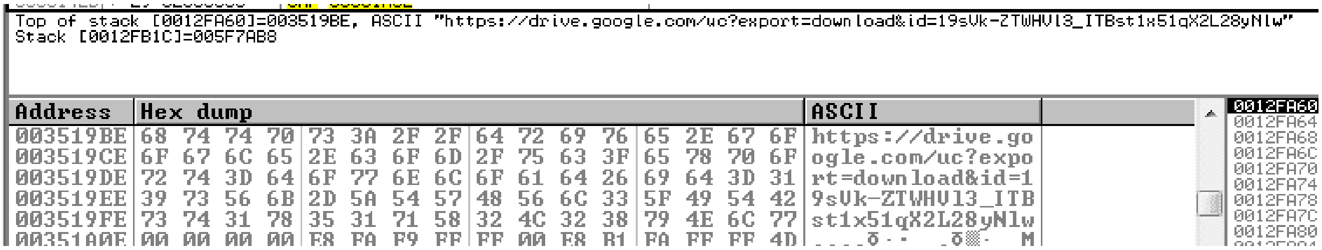
Autorun Entry	Description	Publisher	Image Path
HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce			
☑ STRM	hemipa	WONderware	c:\users\tst\bemeete\rette.exe

Figure 14.0 Achieving malware persistence

Now GuLoader implements process hollowing:


Malware
Created process
1.
GuLoader
Creates a process in suspended state
2.
GuLoader
Mapping of msvbvm60.dll in the created process
3.
GuLoader
Decrypted malicious code
Injects the decrypted malicious code in the created suspended process
4.
GuLoader
Decrypted malicious code
Sets the process context to redirect to the inject malicious code
5.
GuLoader
Decrypted malicious code
Calls NtResumeThread to resume the suspended process and will be executed

The child process (for this sample it’s *RegAsm.exe*) downloads and decrypts the encrypted payload from a cloud hosting service, and maps the decrypted payload into memory to execute.



```
Top of stack [0012FA60]=003519BE, ASCII "https://drive.google.com/uc?export=download&id=19sVk-ZTWHVl3_ITBst1x51qX2L28yNlw"
Stack [0012FB1C]=005F7AB8

Address |Hex dump                                        |ASCII
003519BE|68 74 74 70|73 3A 2F 2F|64 72 69 76|65 2E 67 6F|https://drive.go
003519CE|6F 67 6C 65|2E 63 6F 6D|2F 75 63 3F|65 78 70 6F|ogle.com/uc?expo
003519DE|72 74 3D 64|6F 77 6E 6C|6F 61 64 26|69 64 3D 31|rt=download&id=1
003519EE|39 73 56 6B|2D 5A 54 57|48 56 6C 33|5F 49 54 42|9sVk-ZTWHVl3_ITB
003519FE|73 74 31 78|35 31 71 58|32 4C 32 38|79 4E 6C 77|st1x51qX2L28yNlw
00351A0E|00 00 00 00|E8 FA F9 FF|FF 00 E8 B1|FA FF FF 4D|....è.. .è▒.   M
```

Figure 10.0 The cloud hosting service storing the encrypted payload

The common GuLoader payloads are Formbook, NetWire, Remcos, Lokibot etc.

IOCs:

URLs

- hxxps://onedrive[.]live[.]com/download?cid=1491235303209D1A&resid=1491235303209D1A!109&authkey=ACw2GiM8jfgliBs
- hxxps://drive[.]google[.]com/uc?export=download&id=1EQ7DIlAk9lk2E52DQLELmB02ADqw-62s
- hxxps://drive[.]google[.]com/uc?export=download&id=19sVk-ZTWHVl3_ITBst1x51qX2L28yNlw

Samples

- IMG and ISO Files
  - 466a8de97917fdbc706ccad735ef08a4b049f802d01a03e4f611f75a132e4839
  - 7aadacc7c5bb0c0319f8943d3c65ef2d41d49b1c470210e70e250dd665f167fe
- EXE Files
  - 503f94f00304bc18900c3494f2da5bcb1d8a103a0b15ce00bbdaeb5dfd8d9b7b
  - cbffd8f471de9728610b1edd4519f65399a8e64e46177e1178685ef6b081065b

VIPRE detects and prevents this kind of malware and associated infections.

Analysis by #Farrallel