

Topic :- Guloader Malware analysis

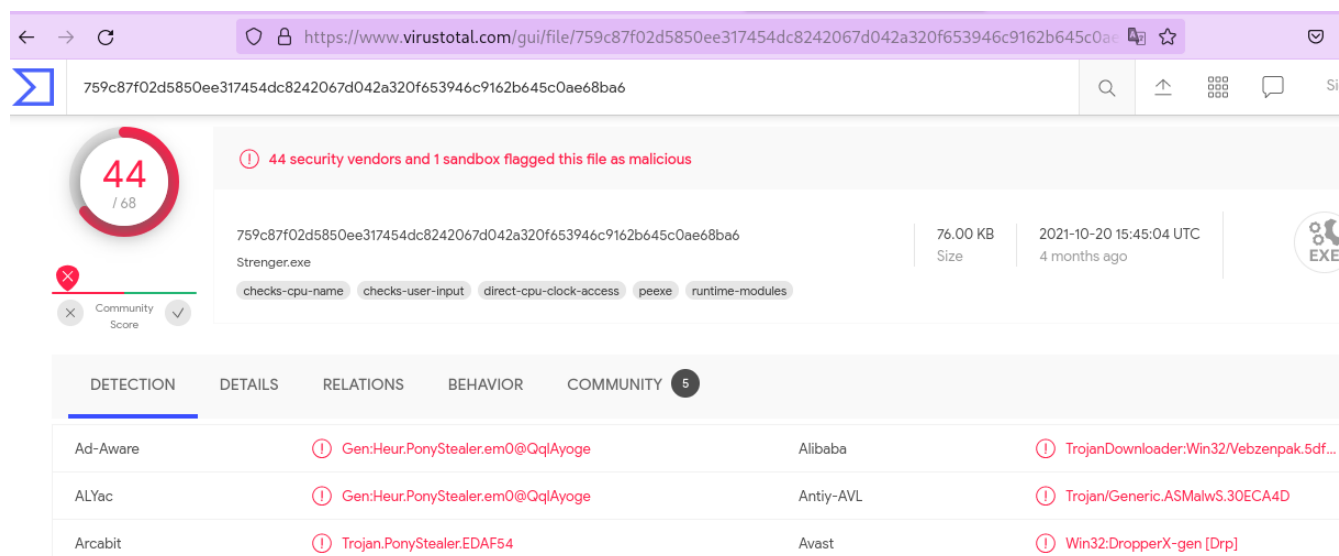
Malware Hash :-

- **sha256** :- 759c87f02d5850ee317454dc8242067d042a320f653946c9162b645c0ae68ba6

Tools :- Windbg, Cutter, online malware sandbox, Process hacker

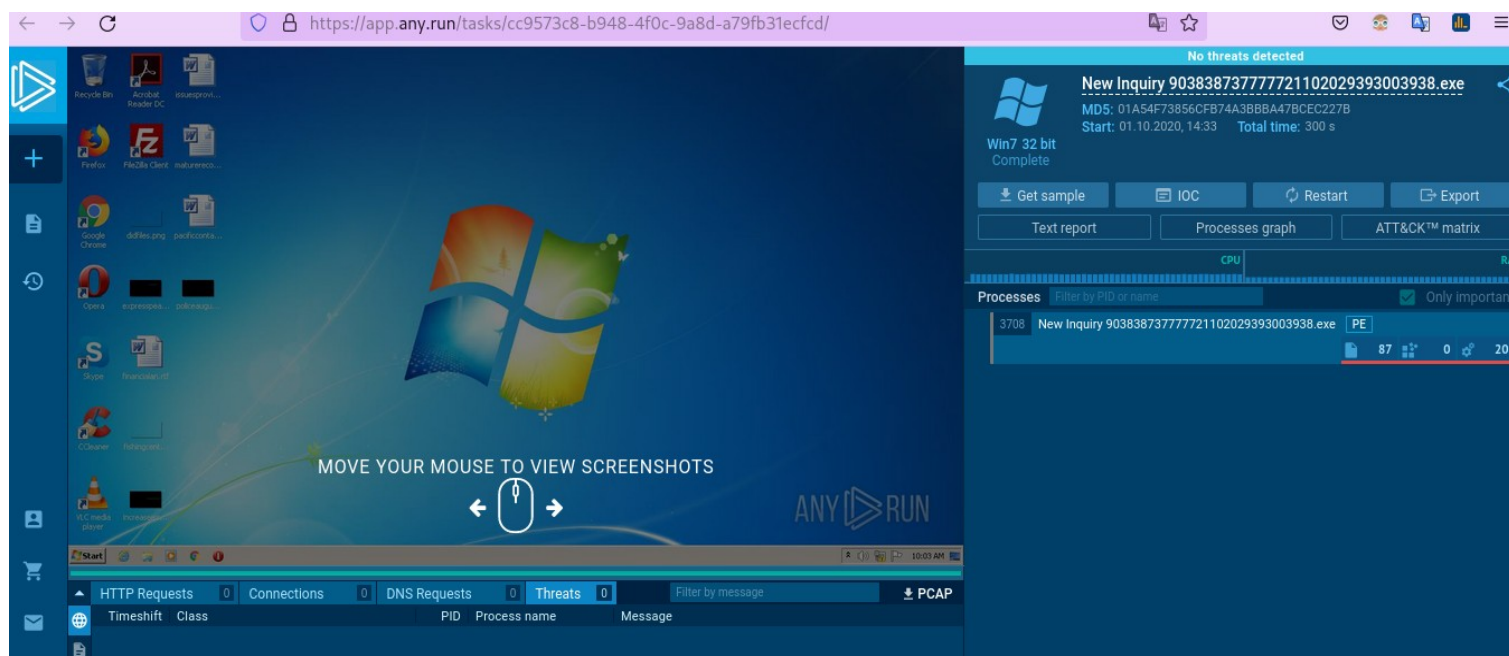
Overview :- Malware downloaders are becoming extremely popular among malicious actor's as it provides a way to gather information about the execution environment. Guloader is popular downloader malware and famous for its anti-debugging and anti-VM capabilities. The evasion tactics make it hard to analyse the malware. In this writeup we analyse the Guloader downloader malware and take a look at the various techniques used by the malware.

We first take a look various online sandbox as to get an overview..



The screenshot shows the VirusTotal web interface. At the top, the URL is <https://www.virustotal.com/gui/file/759c87f02d5850ee317454dc8242067d042a320f653946c9162b645c0ae68ba6>. The file is identified as 'Strenger.exe' (76.00 KB, 2021-10-20 15:45:04 UTC). A large red circle indicates a score of 44/68. A message states: '44 security vendors and 1 sandbox flagged this file as malicious'. Below this, a list of detected threats is shown:

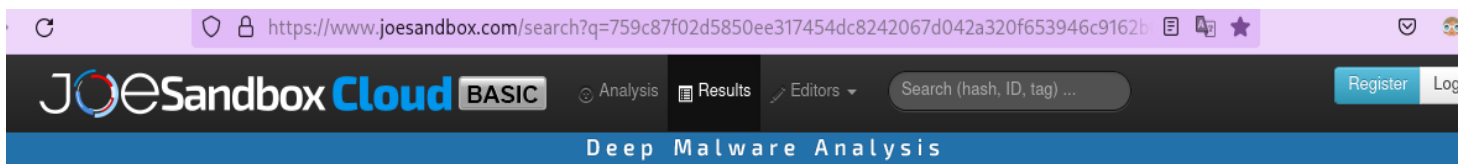
Detection	Details	Relations	Behavior	Community
Ad-Aware	Gen:Heur.PonyStealer.em0@QqIAyoge	Alibaba	TrojanDownloader:Win32/Vebzenpak.5df...	
ALYac	Gen:Heur.PonyStealer.em0@QqIAyoge	Antiy-AVL	Trojan/Generic.ASMalWS.30ECA4D	
Arcabit	Trojan.PonyStealer.EDAF54	Avast	Win32:DropperX-gen [Drp]	



The screenshot shows the app.any.run web interface. The URL is <https://app.any.run/tasks/cc9573c8-b948-4f0c-9a8d-a79fb31ecfcd/>. The file is identified as 'New Inquiry 90383873777721102029393003938.exe' (MD5: 01A54F73856CFB74A38BB8A47BCEC227B, Start: 01.10.2020, 14:33, Total time: 300 s). The interface shows a Windows 7 desktop environment with various icons and a taskbar. On the right side, there is a sidebar with the following information:

- No threats detected
- Win7 32 bit Complete
- Get sample, IOC, Restart, Export buttons
- Text report, Processes graph, ATT&CK™ matrix buttons
- CPU usage bar chart
- Processes list: 3708 New Inquiry 90383873777721102029393003938.exe (PE, 87% CPU, 0% RAM, 20% Disk)
- Threats list: 0
- PCAP button

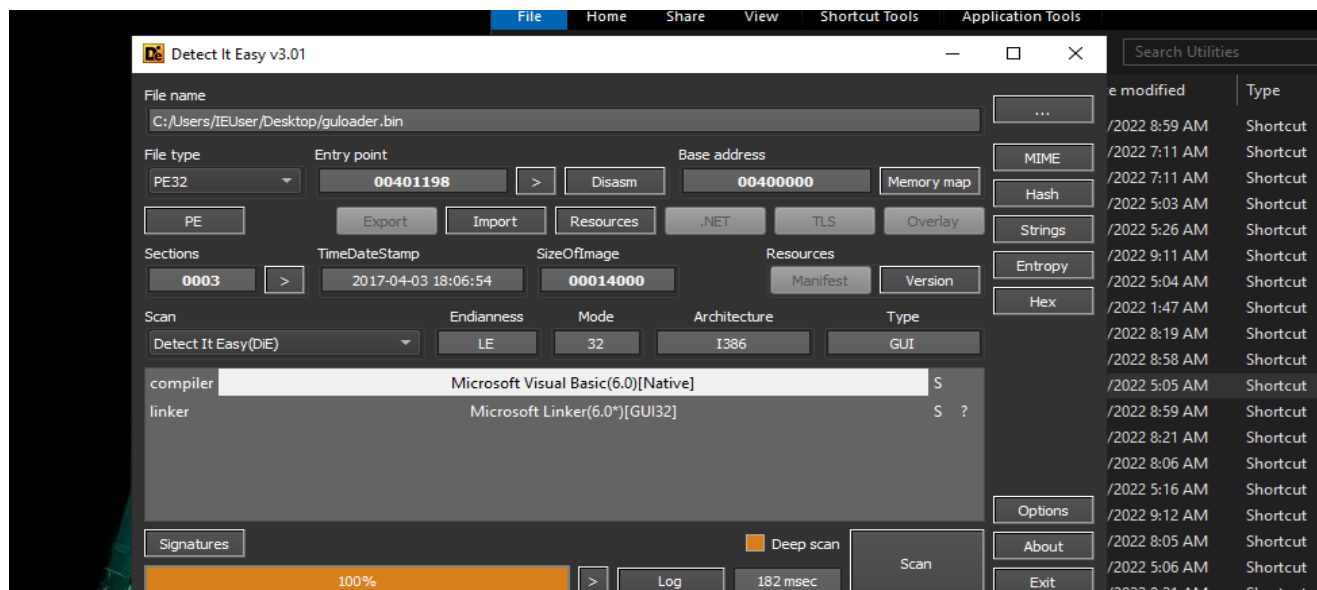
Looking at **app.any.run** we can see that it does not even recognize the file as malicious. On the other hand, **Virustotal** does have more than 44 vendors recognizing it as malware, more precisely as **Trojan Downloader**. If you look closely, we can see that **PonyStealer** is being detected. That is mostly because of similarity code for the malware's. Joe sandbox does detect it as Guloader malware downloader.



Search results for "759c87f02d5850ee317454dc8242067d042a320f653946c9162b645c0ae68ba6"
(limited to max. 20)

Result	Threat	Antivirus	Icon	Time & Date	Name	Info	Class	Graph	Actions
	MALICIOUS	GuLoader Lumino...	21%		17.09.2020 16:30:18	New Inquiry 90383873...			
	MALICIOUS	GuLoader Lumino...	21%		17.09.2020 16:09:28	New Inquiry 90383873...			

Taking a look at the file type we can see that it's a Visual basic file, but later we will see it does extract a shellcode and executes it which we can debug using window's debugger.



The malware allocates memory, decrypts the required shellcode and then directly jumps to it as seen below.

```

Breakpoint 1 hit
KERNELBASE!VirtualAlloc:
765c4d20 8bff      mov     edi,edi
0:000:x86> gu
MSVBVM60!ThunRTMain+0x1ede:
752dcb6f 85c0      test    eax,eax
0:000:x86> g
Breakpoint 1 hit
KERNELBASE!VirtualAlloc:
765c4d20 8bff      mov     edi,edi
0:000:x86> gu
MSVBVM60!ThunRTMain+0x1ede:
752dcb6f 85c0      test    eax,eax
0:000:x86> g
Breakpoint 1 hit
KERNELBASE!VirtualAlloc:
765c4d20 8bff      mov     edi,edi
0:000:x86> gu
MSVBVM60!ThunRTMain+0x1ede:
752dcb6f 85c0      test    eax,eax
0:000:x86> g
ModLoad: 750d0000 75189000 C:\Windows\SysWOW64\textinputframework.dll
ModLoad: 74e50000 750ce000 C:\Windows\SysWOW64\CoreUIComponents.dll
ModLoad: 74db0000 74e4b000 C:\Windows\SysWOW64\CoreMessaging.dll
ModLoad: 77020000 770a7000 C:\Windows\SysWOW64\SHCORE.dll
ModLoad: 74d80000 74da9000 C:\Windows\SysWOW64\ntmarta.dll
ModLoad: 76c40000 76ca3000 C:\Windows\SysWOW64\WS2_32.dll
ModLoad: 74ca0000 74d7b000 C:\Windows\SysWOW64\wintypes.dll
ModLoad: 74c00000 74c94000 C:\Windows\SysWOW64\TextShaping.dll
Breakpoint 1 hit
KERNELBASE!VirtualAlloc:
765c4d20 8bff      mov     edi,edi
0:000:x86>

```

765c4d18	cc	int	3
765c4d19	cc	int	3
765c4d1a	cc	int	3
765c4d1b	cc	int	3
765c4d1c	cc	int	3
765c4d1d	cc	int	3
765c4d1e	cc	int	3
765c4d1f	cc	int	3
KERNELBASE!VirtualAlloc:			
765c4d20	8bff	mov	edi,edi
765c4d22	55	push	ebp
765c4d23	8bec	mov	ebp,esp
765c4d25	51	push	ecx
765c4d26	51	push	ecx
765c4d27	8b450c	mov	eax,dword ptr
765c4d2a	8945f8	mov	dword ptr [ebp

Scratch Pad

Ln 0, Col 0 Sys 0<Local> P

```

752dcb6f 85c0      test    eax,eax
0:000:x86> g
Breakpoint 1 hit
KERNELBASE!VirtualAlloc:
765c4d20 8bff      mov     edi,edi
0:000:x86> gu
MSVBVM60!ThunRTMain+0x1ede:
752dcb6f 85c0      test    eax,eax
0:000:x86> g
ModLoad: 750d0000 75189000 C:\Windows\SysWOW64\textinputframework.dll
ModLoad: 74e50000 750ce000 C:\Windows\SysWOW64\CoreUIComponents.dll
ModLoad: 74db0000 74e4b000 C:\Windows\SysWOW64\CoreMessaging.dll
ModLoad: 77020000 770a7000 C:\Windows\SysWOW64\SHCORE.dll
ModLoad: 74d80000 74da9000 C:\Windows\SysWOW64\ntmarta.dll
ModLoad: 76c40000 76ca3000 C:\Windows\SysWOW64\WS2_32.dll
ModLoad: 74ca0000 74d7b000 C:\Windows\SysWOW64\wintypes.dll
ModLoad: 74c00000 74c94000 C:\Windows\SysWOW64\TextShaping.dll
Breakpoint 1 hit
KERNELBASE!VirtualAlloc:
765c4d20 8bff      mov     edi,edi
0:000:x86> gu
guloader+0x42b1:
004042b1 d9d0      fnop
0:000:x86> t
guloader+0x42b3:
004042b3 51        push    ecx
0:000:x86> bp 404c99
0:000:x86> g
Breakpoint 2 hit
guloader+0x4c99:
00404c99 ffd0      call    eax {004b0000}
0:000:x86>

```

00404c/b	c1e/b6	shl	edi, 0B6h
00404c7e	61	popad	
00404c7f	51	push	ecx
00404c80	21c1	and	ecx,eax
00404c82	59	pop	ecx
00404c83	d9d0	fnop	
00404c85	53	push	ebx
00404c86	81cbf7820da5	or	ebx,0A50D82F7h
00404c8c	5b	pop	ebx
00404c8d	50	push	eax
00404c8e	05d5000000	add	eax,0D5h
00404c93	3d5ff31fa2	cmp	eax,0A21FF35Fh
00404c98	58	pop	eax
00404c99	ffd0	call	eax {004b0000}
00404c9b	60	pushad	
00404c9c	c1ee39	shr	esi,39h
00404c9f	c1e739	shl	edi,39h
00404ca2	61	popad	
00404ca3	53	push	ebx
00404ca4	81cbdd83fce2	or	ebx,0E2FC83DDh
00404caa	5b	pop	ebx
00404cab	51	push	ecx
00404cac	21c1	and	ecx,eax
00404cae	59	pop	ecx
00404caf	50	push	eax
00404cb0	83c05b	add	eax,5Bh
00404cb3	3db86efe50	cmp	eax,50FE6EB8h

Scratch Pad

Ln 0, Col 0 Sys 0<Local> Proc 00

The malware at various points uses hashes for strings to retrieve the desired API. Some of the API's required for important functionality for the malware are given below. The stack address are also given, which will be usefull to understand the disassembly later.

- 60AF076D - kernel32!TerminateProcess
- B76339E - kernel32.ExitProcess
- 82962C8 - NtProtectVirtualMemory
- 321C9581 - ntdll.DbgBreakpoint
- 6F0BDB18 - ntdll.DbgUiRemoteBreakpoint
- EB96C5FA - kernel32.CreateFileA
- 54212E31 - NtSetInformationThread

→ 6793C34C	- ntdll.ZwAllocateVirtualMemory	
→ 9E0E1A44	- ntdll.NtGetContextThread	- [ebp+28]
→ 308BE0D0	- NtSetContextThread	- [ebp+2C]
→ 95F3A792	- ntdll.ZwWriteVirtualMemory	- [ebp+30]
→ D02E20D0	- ntdll.NtCreateSection	- [ebp+38]
→ 231F196A	- ntdll.NtMapViewOfSection	- [ebp+3C]
→ C29C5019	- NtOpenFile	- [ebp+FC]
→ 8B8E133D	- ntdll.ZwClose	- [ebp+40]
→ 2C7B3D30	- NtResumeThread	- [ebp+118]
→ 4DA0ACCC	- kernel32.CreateProcessInternalW	- [ebp+50]
→ E19E5FE	- kernel32.Sleep	- [ebp+BC]
→ 7F08F451	- kernel32.CreateThread	- [ebp+C0]
→ ECCDA1BA	- kernel32.WaitForSingleObject	- [ebp+34]
→ 87AE6A46	- kernel32.TerminateThread	- [ebp+C4]
→ EB96C610	- kernel32.CreatorFileW	- [ebp+78]
→ 663CEC80	- kernel32.WriteFile	- [ebp+B0]
→ 3870CA07	- kernel32.CloseHandle	- [ebp+54]
→ 7891C520	- kernel32.GetFileSize	- [ebp+120]
→ 71019921	- kernel32.ReadFile	- [ebp+124]

The malware uses **<ntdll.ZwProtectVirtualMemory>** to change the rights of the ntdll file to write and executable.

The screenshot shows a debugger window with two panes. The left pane displays a memory dump with addresses from 0019f500 to 0019f504. The right pane displays assembly code from 01f29b44 to 01f29b90. A blue arrow points to the instruction `call eax {ntdll!NtProtectVirtualMemory (77452ed0)}` in the assembly view. Another blue arrow points to the memory address `00000000` in the memory dump view, which is labeled `size`. The assembly code includes instructions like `ret 4`, `cmp dl,dl`, `mov ebx,dword ptr fs:[0C0h]`, `test dh,ch`, `cmp ebx,0`, `je 01f29b9d`, `test bh,ah`, `jmp 01f29b9e`, `cmp cx,ax`, `pop eax`, `cmp dl,bl`, `test bx,ax`, `jmp 01f29b98`, `pop edx`, `mov bx,33h`, `push bx`, and `push eax`.

It then goes on to patch the API's **<ntdll.DebugBreakPoint>** with NOP and **<ntdll.DebugUiRemoteBreakin>** with a call to **<Kernel32.ExitProcess>** API.

```
eax=00000000 ebx=773e1000 ecx=473c0000 edx=00000000
eip=01f29520 esp=0019f4fc ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053
01f29520 8b442418  mov     eax,dword ptr [esp+18h]
0:000> t
eax=77454d30 ebx=773e1000 ecx=473c0000 edx=00000000
eip=01f29524 esp=0019f4fc ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053
01f29524 c60090  mov     byte ptr [eax],90h
0:000> u @eax
ntdll!DbgBreakPoint:
77454d30 cc      int      3
77454d31 c3      ret
77454d32 cc      int      3
77454d33 cc      int      3
77454d34 cc      int      3
77454d35 cc      int      3
77454d36 cc      int      3
77454d37 cc      int      3
Scratch Pad Command
```

```
01f2950e 6a40      push    40h
01f29510 e837060000 call    01f29b4c
01f29515 83f800    cmp     eax,0
01f29518 0f8538050000 jne     01f29a56
01f2951e 84e4      test    ah,ah
01f29520 8b442418  mov     eax,dword ptr [esp+18h]
01f29524 c60090    mov     byte ptr [eax],90h
01f29527 f7c1631f686d test    ecx,6D681F63h
01f2952d 39c1      cmp     ecx,eax
01f2952f 8b44241c  mov     eax,dword ptr [esp+1Ch]
01f29533 c6006a    mov     byte ptr [eax],6Ah
01f29536 c6400100 mov     byte ptr [eax+1],0
01f2953a 6685cb    test    bx,cx
01f2953d c64002b8 mov     byte ptr [eax+2],0B8h
01f29541 38d3      cmp     bl,dl
01f29543 66f7c645e3 test    si,0E345h
01f29548 8b953c010000 mov     edx,dword ptr [ebp+13Ch]
01f2954e 895003    mov     dword ptr [eax+3],edx
01f29551 c64007ff mov     byte ptr [eax+7],0FFh
01f29555 85db      test    ebx,ebx
01f29557 c64008d0 mov     byte ptr [eax+8],0D0h
01f2955b 663df23d cmp     ax,3DF2h
01f2955f c64009c2 mov     byte ptr [eax+9],0C2h
01f29563 c6400a04 mov     byte ptr [eax+0Ah],4
```

Ln 0, Col 0 Sys 0: <Local> Proc 000:2520 Thrd 000:2058 ASM OVR

```
0:000> u @eax
ntdll!DbgBreakPoint:
77454d30 90      nop
77454d31 c3      ret
77454d32 cc      int      3
77454d33 cc      int      3
77454d34 cc      int      3
77454d35 cc      int      3
77454d36 cc      int      3
77454d37 cc      int      3
0:000>
ntdll!DbgBreakPoint:
77454d30 90      nop
77454d31 c3      ret
77454d32 cc      int      3
77454d33 cc      int      3
77454d34 cc      int      3
77454d35 cc      int      3
77454d36 cc      int      3
77454d37 cc      int      3
Scratch Pad Command
```

```
01f2950c 85d8      test    eax,ebx
01f2950e 6a40      push    40h
01f29510 e837060000 call    01f29b4c
01f29515 83f800    cmp     eax,0
01f29518 0f8538050000 jne     01f29a56
01f2951e 84e4      test    ah,ah
01f29520 8b442418  mov     eax,dword ptr [esp+18h]
01f29524 c60090    mov     byte ptr [eax],90h
01f29527 f7c1631f686d test    ecx,6D681F63h
01f2952d 39c1      cmp     ecx,eax
01f2952f 8b44241c  mov     eax,dword ptr [esp+1Ch]
01f29533 c6006a    mov     byte ptr [eax],6Ah
01f29536 c6400100 mov     byte ptr [eax+1],0
01f2953a 6685cb    test    bx,cx
01f2953d c64002b8 mov     byte ptr [eax+2],0B8h
01f29541 38d3      cmp     bl,dl
01f29543 66f7c645e3 test    si,0E345h
01f29548 8b953c010000 mov     edx,dword ptr [ebp+13Ch]
01f2954e 895003    mov     dword ptr [eax+3],edx
01f29551 c64007ff mov     byte ptr [eax+7],0FFh
01f29555 85db      test    ebx,ebx
01f29557 c64008d0 mov     byte ptr [eax+8],0D0h
01f2955b 663df23d cmp     ax,3DF2h
01f2955f c64009c2 mov     byte ptr [eax+9],0C2h
01f29563 c6400a04 mov     byte ptr [eax+0Ah],4
01f29567 6639db    cmp     bx,bx
```

Ln 0, Col 0 Sys 0: <Local> Proc 000:2520 Thrd 000:2058 ASM OVR

```
7748dc00 18a16428 f6000000 000fca80 19752000
7748dc0d 00fc6583 fc7057e8 3307ebff 8bc340c0
7748dce0 45c7e865 fffffefc e8006aff fffbd7c0
7748dcf0 cccccccc cccccccc cccccccc cccccccc
7748dd00 8b55ff8b 0d8b64ec 00000018 8908458b
7748dd10 000f2481 04c25d00 ccccccc0 cccccccc
0:000> u @eax
ntdll!DbgUiRemoteBreakin:
7748dca0 6a08      push    8
7748dca2 6870cb4e77 push    offset ntdll!QueryReg:
7748dca7 e8589ffdf call    ntdll!_SEH_prolog4 (7:
7748dcac 64a130000000 mov     eax,dword ptr fs:[0000
7748dcb2 80780200  cmp     byte ptr [eax+2],0
7748dcb6 jne       ntdll!DbgUiRemoteBrea
7748dcb8 f605d402fe7f02 test    byte ptr [SharedUserDe
7748dcbf 7428      je      ntdll!DbgUiRemoteBrea
Scratch Pad Command
```

```
01f29520 8b442418  mov     eax,dword ptr [esp+18h]
01f29524 c60090    mov     byte ptr [eax],90h
01f29527 f7c1631f686d test    ecx,6D681F63h
01f2952d 39c1      cmp     ecx,eax
01f2952f 8b44241c  mov     eax,dword ptr [esp+1Ch]
01f29533 c6006a    mov     byte ptr [eax],6Ah
01f29536 c6400100 mov     byte ptr [eax+1],0
01f2953a 6685cb    test    bx,cx
01f2953d c64002b8 mov     byte ptr [eax+2],0B8h
01f29541 38d3      cmp     bl,dl
01f29543 66f7c645e3 test    si,0E345h
01f29548 8b953c010000 mov     edx,dword ptr [ebp+13Ch]
01f2954e 895003    mov     dword ptr [eax+3],edx
01f29551 c64007ff mov     byte ptr [eax+7],0FFh
01f29555 85db      test    ebx,ebx
01f29557 c64008d0 mov     byte ptr [eax+8],0D0h
01f2955b 663df23d cmp     ax,3DF2h
01f2955f c64009c2 mov     byte ptr [eax+9],0C2h
01f29563 c6400a04 mov     byte ptr [eax+0Ah],4
01f29567 6639db    cmp     bx,bx
01f2956a c6400b00 mov     byte ptr [eax+0Bh],0
01f2956e 85d9      test    ecx,ecx
01f29570 84e6      test    dh,ah
```

Ln 0, Col 0 Sys 0: <Local> Proc 000:2520 Thrd 000:205


```

eip=01f2956a esp=0019f4fc ebp=0019f51c iopl=0         nv up ei pl zr na pr nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=000f
01f2956a c6400b00      mov     byte ptr [eax+0Bh],0      ds:002b:7748c
0:000> p
eax=7748dca0 ebx=773e1000 ecx=473c0000 edx=77224e10 esi=00000000 edi=01f2
eip=01f2956e esp=0019f4fc ebp=0019f51c iopl=0         nv up ei pl zr na pr nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=000f
01f2956e 85d9          test    ecx,ebx
0:000> u @eax
ntdll!DbgUiRemoteBreakin:
7748dca0 6a00      push    0
7748dca2 b8104e2277 mov     eax,offset KERNEL32!ExitProcessImplement
7748dca7 ffd0      call    eax
7748dca9 c20400    ret     4
7748dcac 64a130000000 mov     eax,dword ptr fs:[00000030h]
7748dcb2 80780200  cmp     byte ptr [eax+2],0
7748dcb6 7509      jne     ntdll!DbgUiRemoteBreakin+0x21 (7748dcc1)
7748dcb8 f605d402fe7f02 test    byte ptr [SharedUserData+0x2d4 (7ffe02d4)

```

```

01f29557 c64008d0      mov     byte ptr [eax+8],0D0h
01f2955b 663df23d      cmp     ax,3DF2h
01f2955f c64009c2      mov     byte ptr [eax+9],0C2h
01f29563 c6400a04      mov     byte ptr [eax+0Ah],4
01f29567 6639db        cmp     bx,bx
01f2956a c6400b00      mov     byte ptr [eax+0Bh],0
01f2956e 85d9          test    ecx,ebx
01f29570 84e6          test    dh,ah
01f29572 85d8          test    eax,ebx
01f29574 89d8          mov     eax,ebx
01f29576 03442408      add     eax,dword ptr [esp+8]
01f2957a 43           inc     ebx
01f2957b 39c3          cmp     ebx,eax
01f2957d 0f84d8040000 je      01f29a5b
01f29583 803bb8        cmp     byte ptr [ebx],0B8h
01f29586 75f2          jne     01f2957a
01f29588 837b0100      cmp     dword ptr [ebx+1],0
01f2958c 75ec          jne     01f2957a
01f2958e 84e4          test    ah,ah
01f29590 807b05ba      cmp     byte ptr [ebx+5],0BAh
01f29594 75e4          jne     01f2957a
01f29596 8b5306        mov     edx,dword ptr [ebx+6]
01f29599 f7c16232598d test    ecx,8D593262h
01f2959f 83c30a        add     ebx,0Ah

```

The malware uses **rdtsc** and **cpuid** for anti-debugging and anti-vm. The usage of **rdtsc** is simple, being called twice and then the difference between the two timings is used to judge if the malware is running inside a debugger.

```

eip=01ef476d esp=0019f510 ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef476d e832040000      call    01ef4ba4
0:000> t
eax=cccccccc ebx=77830033 ecx=000186a0 edx=01ef9b9d e
eip=01ef4ba4 esp=0019f50c ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef4ba4 0faee8          lfence
0:000> t
eax=cccccccc ebx=77830033 ecx=000186a0 edx=01ef9b9d e
eip=01ef4ba7 esp=0019f50c ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef4ba7 0f31          rdtsc
0:000> t
eax=2712f1cb ebx=77830033 ecx=000186a0 edx=00000238 e
eip=01ef4ba9 esp=0019f50c ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef4ba9 0faee8          lfence
0:000> t
eax=2712f1cb ebx=77830033 ecx=000186a0 edx=00000238 e
eip=01ef4bac esp=0019f50c ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef4bac c1e220          shl     edx,20h

```

```

01ef4b84 d113          mov     cl,13h
01ef4b84 ba25b113ba      mov     edx,0BA13B125h
01ef4b89 25b113ba25      and     eax,25BA13B1h
01ef4b8e b113          mov     cl,13h
01ef4b90 ba25b113ba      mov     edx,0BA13B125h
01ef4b95 25b11338f6      and     eax,0F63813B1h
01ef4b9a c3           ret
01ef4b9b 66f7c1b0bc      test    cx,0BCB0h
01ef4ba0 85c0          test    eax,eax
01ef4ba2 3c23          cmp     al,23h
01ef4ba4 0faee8          lfence
01ef4ba7 0f31          rdtsc
01ef4ba9 0faee8          lfence
01ef4bac c1e220          shl     edx,20h
01ef4baf 09c2          or      edx,eax
01ef4bb1 c3           ret
01ef4bb2 6681fef7db      cmp     si,0DBF7h
01ef4bb7 e88edfffff      call    01ef2b4a
01ef4bbc e804440000      call    01ef8fc5
01ef4bcb e8e4f9ffff      call    01ef45aa
01ef4bc6 8b580c          mov     ebx,dword ptr [eax+0Ch]
01ef4bc9 395808          cmp     dword ptr [eax+8],ebx
01ef4bcc 7d0d          jge     01ef4bdb

```

```

01ef4ba9 0faee8          lfence
0:000> t
eax=2712f1cb ebx=77830033 ecx=000186a0 edx=00000238 e
eip=01ef4bac esp=0019f50c ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef4bac c1e220          shl     edx,20h
0:000> t
eax=2712f1cb ebx=77830033 ecx=000186a0 edx=00000238 e
eip=01ef4baf esp=0019f50c ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef4baf 09c2          or      edx,eax
0:000> t
eax=2712f1cb ebx=77830033 ecx=000186a0 edx=2712f3fb e
eip=01ef4bb1 esp=0019f50c ebp=0019f51c iopl=0
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
01ef4bb1 c3           ret
0:000> t
eax=2712f1cb ebx=77830033 ecx=000186a0 edx=2712f3fb e
eip=01ef4772 esp=0019f510 ebp=0019f51c iopl=0

```

```

01ef474c 031700          cmp     ecx,0
01ef474f 75de          jne     01ef472f
01ef4751 81bd9c00000060ea0000 cmp     dword ptr [ebp+9Ch],0EA60
01ef475b 7fb6          jg      01ef4713
01ef475d 83ff00          cmp     edi,0
01ef4760 7cb1          jl      01ef4713
01ef4762 81ff80778e06      cmp     edi,68E7780h
01ef4768 7da9          jge     01ef4713
01ef476a 89f8          mov     eax,edi
01ef476c c3           ret
01ef476d e832040000      call    01ef4ba4
01ef4772 89d6          mov     esi,edx
01ef4774 60          pushed  eax
01ef4775 b801000000      mov     eax,1
01ef477a 0fa2          cpuid
01ef477c 0fbae11f        bt      ecx,1Fh
01ef4780 0f82fe420000      jnb     01ef8a84
01ef4786 61          popad
01ef4787 e818040000      call    01ef4ba4

```

As can be seen above, **CPUID** is used with a value of 1 being passed to **eax**. This returns CPU feature information in **ecx** register. The 31st bit of the 32 bit value returned determines if the environment is a hypervisor or not. If inside a hypervisor, the malware process ends.

25	sse	SSE instructions (a.k.a. Katmai New Instructions)	aes	AES instruction set
26	sse2	SSE2 instructions	xsave	XSAVE, XRESTOR, XSETBV, XGETBV
27	ss	CPU cache implements self-snoop	osxsave	XSAVE enabled by OS
28	htt	Hyper-threading	avx	Advanced Vector Extensions
29	tm	Thermal monitor automatically limits temperature	f16c	F16C (half-precision) FP feature
30	ia64	IA64 processor emulating x86	rdrnd	RDRAND (on-chip random number generator) feature
31	pbe	Pending Break Enable (PBE# pin) wakeup capability	hypervisor	Hypervisor present (always zero on physical CPUs) ^{[12][13]}

Reserved fields should be masked before using them for processor identification purposes.

EAX=2: Cache and TLB Descriptor information [\[edit \]](#)

Then the malware uses the **EnumWindows** API. The EnumWindows API enumerates all the process and executes a callback function for each process. In this case, it just increments a counter and later compares it with 0xc. Depending on the value, the malware proceeds or exits.

```

SD 00f/C0e00C      test     SI, 0CEDh
t
08bdd0 ebx=01f20613 ecx=0000010a edx=00000000 esi=
f205f0 esp=0019f518 ebp=0019f51c iopl=0         01f205e5 5b          pop     ebx
3  ss=002b  ds=002b  es=002b  fs=0053  gs=002b   01f205e6 84ff       test    bh,bh
f0 54          push    esp                    01f205e8 31d2       xor     edx,edx
08bdd0 ebx=01f20613 ecx=0000010a edx=00000000 esi=  01f205ea 52          push    edx
f205f1 esp=0019f514 ebp=0019f51c iopl=0         01f205eb 66f7c6ed8c test    si,8CEDh
3  ss=002b  ds=002b  es=002b  fs=0053  gs=002b   01f205f0 54          push    esp
f1 39c0        cmp     eax,eax                    01f205f1 39c0        cmp     eax,eax
08bdd0 ebx=01f20613 ecx=0000010a edx=00000000 esi=  01f205f3 53          push    ebx
f205f3 esp=0019f514 ebp=0019f51c iopl=0         01f205f4 ffd0       call    eax {USER32!EnumWindows (7708bdd0)}
3  ss=002b  ds=002b  es=002b  fs=0053  gs=002b   01f205f6 58          pop     eax
f3 53          push    ebx                    01f205f7 6639d2      cmp     dx,dx
08bdd0 ebx=01f20613 ecx=0000010a edx=00000000 esi=  01f205fa 83f80c      cmp     eax,0Ch
f205f4 esp=0019f510 ebp=0019f51c iopl=0         01f205fd 7d2d       jge     01f2062c
3  ss=002b  ds=002b  es=002b  fs=0053  gs=002b   01f205ff 6a00       push    0
f3 53          push    0FFFFFFFFh            01f20601 6aff       push    0FFFFFFFFh
f4 ffd0       call    eax {USER32!EnumWindows}  01f20603 ff9598000000 call   dword ptr [ebp+98h]
08bdd0 ebx=01f20613 ecx=0000010a edx=00000000 esi=  01f20609 84d9       test    cl,bl
f205f4 esp=0019f510 ebp=0019f51c iopl=0         01f2060b f6c3ed      test    bl,0EDh
3  ss=002b  ds=002b  es=002b  fs=0053  gs=002b   01f2060e e8d2ffff    call    01f205e5
f4 ffd0       call    eax {USER32!EnumWindows}  01f20613 8b4c2408    mov     ecx,dword ptr [esp+8]
08bdd0 ebx=01f20613 ecx=0000010a edx=00000000 esi=  01f20617 8b01       mov     eax,dword ptr [ecx]
f205f4 esp=0019f510 ebp=0019f51c iopl=0         01f20619 6639cb      cmp     bx,cx
3  ss=002b  ds=002b  es=002b  fs=0053  gs=002b   01f2061c 40          inc     eax
f4 ffd0       call    eax {USER32!EnumWindows}  01f2061d 8901       mov     dword ptr [ecx],eax
08bdd0 ebx=01f20613 ecx=0000010a edx=00000000 esi=  01f2061f 38d3       cmp     bl,dl
f205f4 esp=0019f510 ebp=0019f51c iopl=0
3  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
f4 ffd0       call    eax {USER32!EnumWindows}

```

Guloder then uses another anti-vm trick, it tries to open files '**C:\Program files\Qemu-ga\qemu-ga.exe**' and '**C:\Program Files\qga\qga.exe**' to check if the malware is running under Virtual Machine or not.

0019f504	00000003	00000000	00000000	01f24c54					
0019f514	01f24cc4	01f20a9b	0019f768	77200000					
0019f524	77223130	7722bd0d	00095f90	00092d58					
0019f534	01f25f68	01f25f5d	0019f668	77452ed0					
0019f544	0019f508	00000000	0019f6a8	7745ad40					
0019f554	de8676bb	00000000	0019f6b8	01f20000					
0019f564	00000000	a9d1531b	027b31e0	02960000					
0:000>	dc	1f24cc4							
01f24cc4	505c3a43	72676f72	46206d61	73656c69					
01f24cd4	6d65515c	61672d75	6d65715c	61672d75					
01f24ce4	6578652e	b4ff8100	667251e5	caf9c7f7					
01f24cf4	ffe295e8	e58955ff	39c3f760	8b185536					
01f24d04	c0840845	3966008b	18588bc3	3881ef38					
01f24d14	c0000005	0472850f	f4380000	83e3c7f6					
01f24d24	840f00fb	00000464	43c7f766	e48453b0					
01f24d34	fff7f2e8	38c389ff	e8d939ff	ffffff86					

01f24c88	6a03	push	3	
01f24c8a	6a00	push	0	
01f24c8c	6a01	push	1	
01f24c8e	6800000080	push	80000000h	
01f24c93	51	push	ecx	
01f24c94	ff959c000000	call	dword ptr [ebp+9Ch] ss:002b:0	
01f24c9a	c20400	ret	4	
01f24c9d	e8c4ffffff	call	01f24c66	
01f24ca2	43	inc	ebx	
01f24ca3	3a5c5072	cmp	bl,byte ptr [eax+edx*2+72h]	
01f24ca7	6f	outs	dx,dword ptr [esi]	
01f24ca8	677261	jb	01f24d0c	
01f24cab	6d	ins	dword ptr es:[edi],dx	
01f24cac	204669	and	byte ptr [esi+69h],al	
01f24caf	6c	ins	byte ptr es:[edi],dx	
01f24cb0	65735c	jae	01f24d0f	
01f24cb3	7167	jno	01f24d1c	
01f24cb5	61	popad		
01f24cb6	5c	pop	esp	
01f24cb7	7167	jno	01f24d20	
01f24cb9	61	popad		
01f24cba	2e	???		

0019f524	77223130	7722bd0d	00095f90	00092d58					
0019f534	01f25f68	01f25f5d	0019f668	77452ed0					
0019f544	0019f508	00000000	0019f6a8	7745ad40					
0019f554	de8676bb	00000000	0019f6b8	01f20000					
0019f564	00000000	a9d1531b	027b31e0	02960000					
0:000>	dc	1f24ca2							
01f24ca2	505c3a43	72676f72	46206d61	73656c69					
01f24cb2	6167715c	6167715c	6578652e	ff87e800					
01f24cc2	3a43ffff	6f72505c	6d617267	6c694620					
01f24cd2	515c7365	2d756d65	715c6167	2d756d65					
01f24ce2	652e6167	81006578	51e5b4ff	c7f76672					
01f24cf2	95e8caf9	55ffffe2	f760e589	553639c3					
01f24d02	08458b18	008bc084	8bc33966	ef381858					
01f24d12	00053881	850fc000	00000472	c7f6f438					

01f24c8c	6a01	push	1	
01f24c8e	6800000080	push	80000000h	
01f24c93	51	push	ecx	
01f24c94	ff959c000000	call	dword ptr [ebp+9Ch] ss:002b:019f5b8+	
01f24c9a	c20400	ret	4	
01f24c9d	e8c4ffffff	call	01f24c66	
01f24ca2	43	inc	ebx	
01f24ca3	3a5c5072	cmp	bl,byte ptr [eax+edx*2+72h]	
01f24ca7	6f	outs	dx,dword ptr [esi]	
01f24ca8	677261	jb	01f24d0c	
01f24cab	6d	ins	dword ptr es:[edi],dx	
01f24cac	204669	and	byte ptr [esi+69h],al	
01f24caf	6c	ins	byte ptr es:[edi],dx	
01f24cb0	65735c	jae	01f24d0f	
01f24cb3	7167	jno	01f24d1c	
01f24cb5	61	popad		
01f24cb6	5c	pop	esp	
01f24cb7	7167	jno	01f24d20	
01f24cb9	61	popad		
01f24cba	2e	???		

The API `<ntdll.NtSetInformationThread>` is then used to hide the thread as seen below.

0x01f20a96	call fcn.01f24c36_anti_vm ; checks for Qemu files
0x01f20a9b	cmp ah, bh
0x01f20a9d	cmp bh, ch
0x01f20a9f	cmp edx, ebx
0x01f20aa1	mov ecx, dword [ebp+0x1c]
0x01f20aa4	mov edx, 0x54212e31 ; '1.IT' ; NtSetInformationThread
0x01f20aa9	call fcn.01f281cf ; retrieve address NtSetInformationThread()
0x01f20aae	mov dword [ebp + 0x130], eax
0x01f20ab4	test bh, bh
0x01f20ab6	test si, 0x8c8f
0x01f20abb	cmp eax, eax
0x01f20abd	push 0
0x01f20abf	push 0
0x01f20ac1	push 0x11 ; 17
0x01f20ac3	cmp dx, dx
0x01f20ac6	push 0xffffffffffffffff
0x01f20ac8	call eax ; call NtSetInformationThread()
0x01f20aca	test al, bl

The **<ntdll.NtSetInformationThread>** API here is called with an undocumented value of 0x11 for the second parameter **ThreadInformationClass**. This will cause the main thread to be hidden, and no information will be passed to the debugger. If any breakpoint is placed in code inside this thread, the debugger will get stuck. The **function fcn.01f281cf** retrieves the address of the API **NtSetInformationThread** using the hash value 0x54212e31. The function is repeatedly used to retrieve addresses of API's. The method to retrieve the address of the API involves using the Export Address Table of the PE file. For more information on dynamic API address resolution, please refer to my previous write-up [here](#).

The malware then gets a handle for the service manager using **ADVAPI32.OpenSCManagerA** API and goes on to check for services related to Virtual Machines using **EnumServiceStatusA** API.

```

fcn.01f211f0
fcn.01f216bf
fcn.01f21719
fcn.01f217bf
fcn.01f217ce
fcn.01f21844
fcn.01f21864
fcn.01f2186b
fcn.01f21d3f
fcn.01f21d5a
fcn.01f21dbc
fcn.01f21e6c
fcn.01f22a91
fcn.01f20b58 test dl, cl
fcn.01f20b5a mov ecx, dword [ebp + 0x5c]
fcn.01f20b5d test ebx, ecx
fcn.01f20b5f cmp ebx, ebx
fcn.01f20b61 mov edx, 0xbaef4789
fcn.01f20b66 test di, 0xe1c4
fcn.01f20b6b call fcn.01f281cf ; retrieve address of ADVAPI32!OpenSCManagerA
fcn.01f20b70 push 4
fcn.01f20b72 push 0
fcn.01f20b74 cmp bl, bl
fcn.01f20b76 test cl, dl
fcn.01f20b78 push 0
fcn.01f20b7a push eax
fcn.01f20b7b call fcn.01f29eee ; anti debug and calls the api whose address is retrieved above
fcn.01f20b80 test dl, cl

```

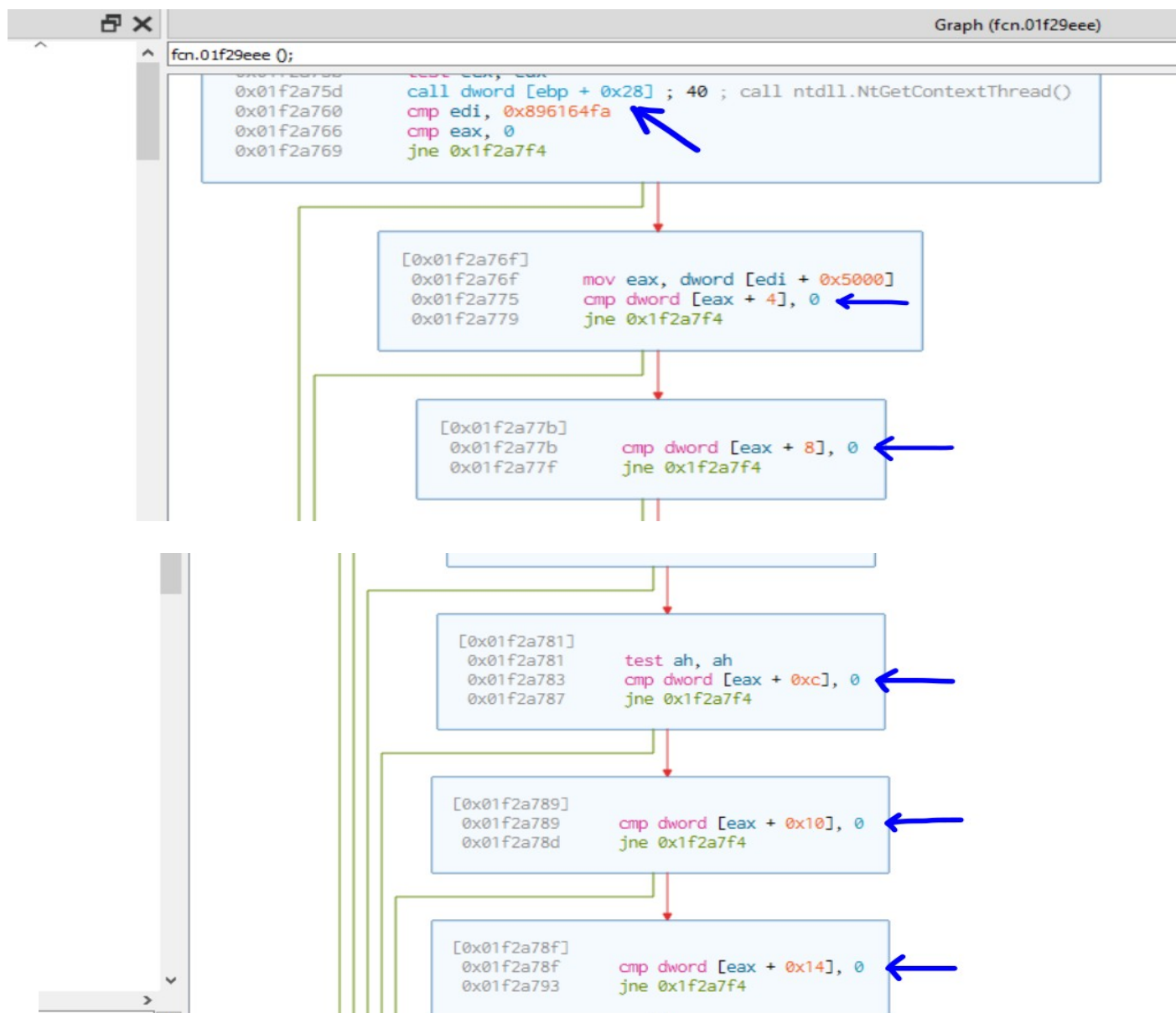
```

d
[0x01f20b8b]
0x01f20b8b test ebx, ecx
0x01f20b8d cmp ebx, ebx
0x01f20b8f mov dword [ebp + 0x9c], eax
0x01f20b95 test di, 0xb188
0x01f20b9a mov ecx, dword [ebp + 0x5c]
0x01f20b9d mov edx, 0x19c2c923
0x01f20ba2 call fcn.01f281cf ; retrieve address of ADVAPI32!EnumServiceStatusA()
0x01f20ba7 cmp bl, bl
0x01f20ba9 test cl, dl
0x01f20bab mov edi, dword [ebp + 0x20]
0x01f20bae add edi, 0x10000
0x01f20bb4 push edi
0x01f20bb5 test dl, cl
0x01f20bb7 add edi, 4
0x01f20bba test ebx, ecx
0x01f20bbc push edi
0x01f20bbd cmp ebx, ebx
0x01f20bbf add edi, 4

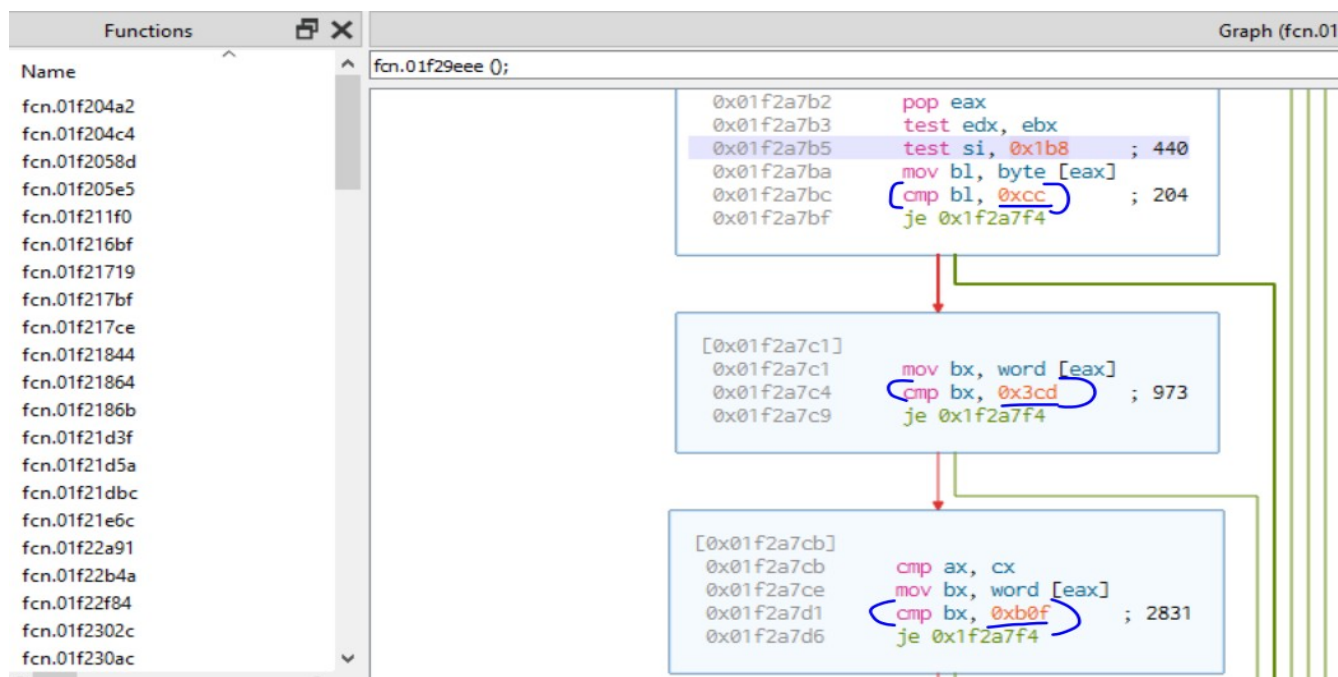
```

The malware before calling the actual Windows API first retrieves the address from the hash of the API using the function **fcn.01f281cf**. It then calls the function **fcn.01f29eee** which calls the required windows API. This pattern is repeated most of the time. The function **fcn.01f29eee** does anti-debug checks before calling the API.

Function **fcn.1f29eee** first uses the **GetContextthread** API to get the thread context which contains the values of the debug registers DR1, DR2 etc. The value of these registers are then compared to 0 to check if hardware breakpoints are setup.



After **hardware breakpoints**, the malware checks for **software breakpoints** by comparing the starting address of the API with **bytes 0xCC, 0xCD, 0xBF** as seen below.



After doing away with all the necessary Anti-VM and Anti-Debugging checks and retrieving the address of the required APIs it does move to the process injection phase. The Process Injection Technique basically involves the following APIs.

- kernel32.CreateProcessInternalW
- ntdll.NtCreateSection
- ntdll.NtMapViewOfSection
- ntdll.ZwAllocateVirtualMemory
- ntdll.ZwWriteVirtualMemory
- ntdll.NtGetContextThread
- ntdll.NtSetContextThread
- ntdll.NtResumeThread

This version of Guloader malware uses the executable file namely "**C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe**" for process creation. The **windir** path is derived using the **PEB**, the address of PEB is derived using **Thread Environment Block** located at **fs:[18]**. The instruction **mov eax, dword [eax+48]** moves the pointer to the environment variables to register **eax**. This pointer is then used to retrieve the **address of windir** which is C:\Windows. The rest of the string is retrieved from the malware binary itself.

Type flag name or address here

Graph (fcn.01f27c58)

fcn.01f27c58 (int32_t arg_4h_2, int32_t arg_4h);

```

[0x01f27c58]
140: fcn.01f27c58 (int32_t arg_4h_2, int32_t arg_4h);
; var int32_t var_4h @ esp+0x18
; arg int32_t arg_4h_2 @ esp+0x1c
; arg int32_t arg_4h @ esp+0x20
0x01f27c58 test dh, bh
0x01f27c5a mov eax, dword fs:[0x18]
0x01f27c60 test dx, dx
0x01f27c63 mov eax, dword [eax + 0x30]
0x01f27c66 mov eax, dword [eax + 0x10]
0x01f27c69 test ah, bh
0x01f27c6b mov eax, dword [eax + 0x48]
0x01f27c6e cmp edx, eax
0x01f27c70 sub eax, 2
0x01f27c73 cmp ax, dx
0x01f27c76 mov esi, dword [arg_4h]
0x01f27c7a cmp ecx, ecx
  
```

The malware first create a process using **<Kernel32.CreateProcessInternalW>** API setting the file parameter as '**C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe**'. It then obtains the handle to the file **C:\Windows.syswow64\msvbvm60.dll** using the **Kernel32.CreateFileA** API and passes the handle to the **ntdll.NtCreateSection**. The **ntdll.NtMapViewOfSection** API is used to map this section to the RegAsm.exe process created previously.

Functions

Name

- fcn.01f204a2
- fcn.01f204c4
- fcn.01f2058d
- fcn.01f205e5
- fcn.01f211f0
- fcn.01f216bf
- fcn.01f21719
- fcn.01f217bf
- fcn.01f217ce
- fcn.01f21844
- fcn.01f21864
- fcn.01f2186b

Graph (fcn.01f29eee)

fcn.01f29eee ();

```

0x01f23756 push dword [ebp + 0x38]
0x01f23759 cmp dh, 0x51 ; 81
0x01f2375c call fcn.01f29eee ; calls ntdll.NtCreateSection()
0x01f23761 test bh, 0xbc ; 188
0x01f23764 cmp eax, 0
0x01f23767 jne 0x1f23df6
  
```

[0x01f2376d]

```

0x01f2376d cmp bl, bl
0x01f2376f cmp ah, dh
0x01f23771 mov edi, dword [ebp + 0x20]
  
```



```

fcf.01f21719
fcf.01f217bf
fcf.01f217ce
fcf.01f21844
fcf.01f21864
fcf.01f2186b
fcf.01f21d3f
fcf.01f21d5a
fcf.01f21dbc
fcf.01f21e6c
fcf.01f22a91
fcf.01f22b4a
fcf.01f22f84
fcf.01f2302c
fcf.01f230ac

```

```

0x01f23798 test edi, 0xa5c75d29
0x01f2379e mov eax, ebp
0x01f237a0 add eax, 0x104 ; 260
0x01f237a5 mov dword [eax], 0x400000
0x01f237ab test cx, 0x60b9
0x01f237b0 push eax
0x01f237b1 push dword [edi + 0x800]
0x01f237b7 push dword [ebp + 0x108]
0x01f237bd cmp dl, bl
0x01f237bf push dword [ebp + 0x3c]
0x01f237c2 call fcf.01f29eee ; calls ntdll.NtMapViewOfSection()
0x01f237c7 cmp eax, 0xc0000018
0x01f237cc je 0x1f23df6

```

Processes	Services	General	Status	Performance	Threads	Token	Modules	Memory	Environment
Name		Handles	Job	GPU	Disk and Network	Comment			
svchost.exe		Hide unnamed handles							
spoolsv.exe		Type	Name	Handle					
svchost.exe		Process	Non-existent process (5396)	0x2b0					
svchost.exe		Process	Non-existent process (3424)	0x2b4					
MsMpEng.exe		Process	RegAsm.exe (3332)	0x2c4					
wlms.exe		Section	\Windows\Theme1827393545	0x1c4					
svchost.exe		Section	\Sessions\1\Windows\Theme1788554724	0x1c8					
svchost.exe		Section	C:\Windows\Fonts\StaticCache.dat	0x264					
NisSrv.exe		Section	C:\Windows\SysWOW64\msvbvm60.dll	0x2d0					
SearchIndexer.exe		Section	\BaseNamedObjects\C:\ProgramData\Microsoft\Windows\Caches*c...	0x460					
SgrmBroker.exe		Section	\BaseNamedObjects\C:\ProgramData\Microsoft\Windows\Caches*c...	0x464					
svchost.exe		Section	\BaseNamedObjects\C:\ProgramData\Microsoft\Windows\Caches*c...	0x468					
svchost.exe		Section	\BaseNamedObjects\C:\ProgramData\Microsoft\Windows\Caches*c...	0x46c					
svchost.exe		Section	\BaseNamedObjects\C:\ProgramData\Microsoft\Windows\Caches*c...	0x744					
ctfmon.exe		Section	\Sessions\1\BaseNamedObjects\SessionImmersiveColorPreference	0xb30					
explorer.exe		Semaphore	\Sessions\1\BaseNamedObjects\C:\?USERS\BOND008\DESKTOP\GUL...	0x184					
VBoxTray.exe		Semaphore	\Sessions\1\BaseNamedObjects\SM0:6048:168:WilStaging_02_p0	0x1cc					
windbg.exe		Thread	guloder.exe (6048): 5732	0x1ec					
guloder.exe		Thread	guloder.exe (6048): 3244	0x26c					
RegAsm.exe		Thread	Non-existent process (5396): 3276	0x2a0					
ProcessHack.exe		Thread	guloder.exe (6048): 4468	0x2a8					
		Thread	windbg.exe (2344): 1672	0x2ac					

In the images above, we can see a pattern where the address of the API is pushed on to the stack and the function fcf.01f29eee is called. This function does anti debugging as previously discussed and calls the required API. The malware then uses the **ZwAllocateVirtualMemory** and **ZwWriteVirtualMemory** to allocate memory inside the process and write shellcode to that memory.

```

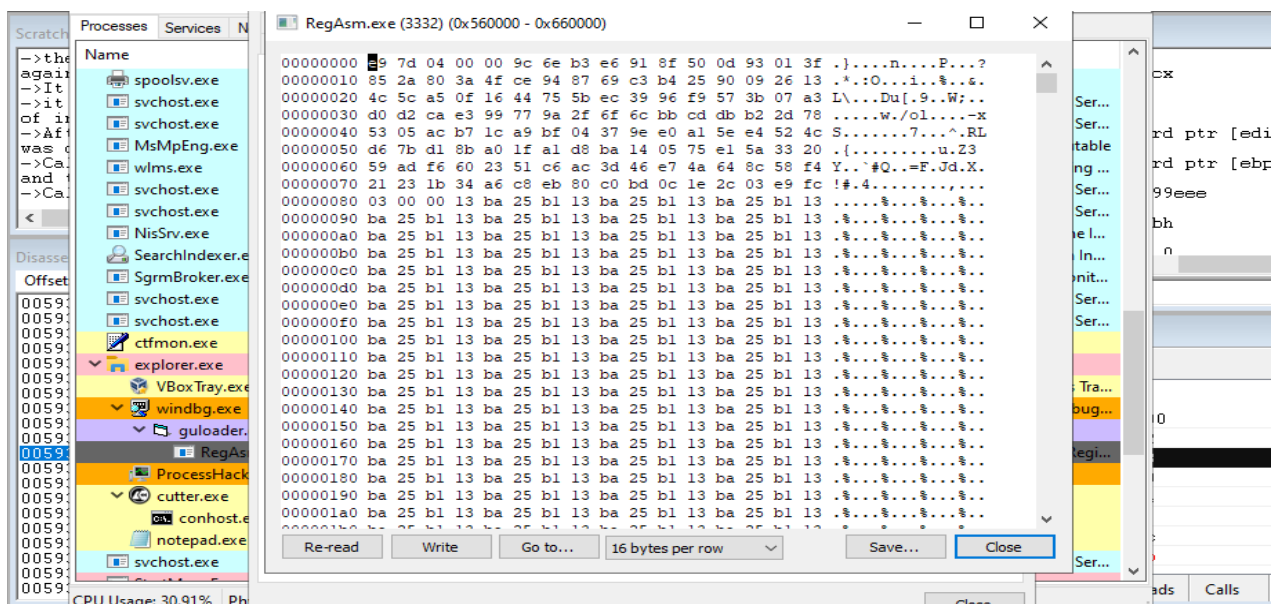
fcf.01f29eee ();

```

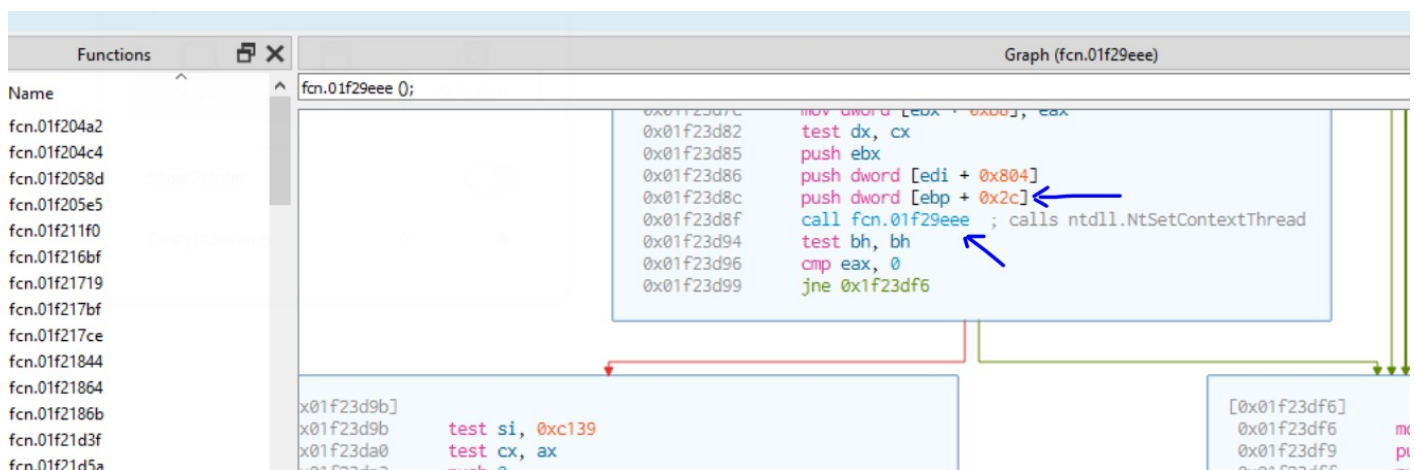
```

0x01f238f2 call 0x1f238d6
0x01f238f7 pop esi
0x01f238f8 cmp eax, 0x70c3c1a3
0x01f238fd push eax
0x01f238fe test ebx, ebx
0x01f23900 cmp bl, bl
0x01f23902 push dword [ebp + 0x44]
0x01f23905 cmp ah, dh
0x01f23907 push dword [ebp + 0x104]
0x01f2390d push dword [edi + 0x800]
0x01f23913 call dword [ebp + 0x30] ; 48 ; call ntdll.NtZwWriteVirtualMemory()
0x01f23916 cmp eax, 0
0x01f23919 jne 0x1f23df6

```



<ntdll.NtGetContextThread> and <ntdll.NtSetContextThread> APIs are used to change the entry point of the process to that of the start of the shellcode in the process. Finally, <ntdll.NtResumeThread> API is used to resume the suspended process.



```

[0x01f23d9b] test si, 0xc139
0x01f23da0 test cx, ax
0x01f23da3 push 0
0x01f23da5 push dword [edi + 0x804]
0x01f23dab push dword [ebp + 0x118]
0x01f23db1 call fcn.01f29eee ; calls ntdll.NtResumeThread()
0x01f23db6 push 0xffff
0x01f23dbb push dword [edi + 0x804]
0x01f23dc1 push dword [ebp + 0x134]
0x01f23dc7 cmp ah, bh
0x01f23dc9 .string "\xe8 a" ; len=4
0x01f23dcd add byte [ebp - 0x2c7a9928], al
0x01f23dd3 ret

[0x01f23df6] mov edi, dword [ebp +
0x01f23df9 push dword [ebp + 0x10
0x01f23dff push dword [ebp + 0x40
0x01f23e02 call fcn.01f29eee
0x01f23e07 push dword [ebp + 0x10
0x01f23e0d push dword [ebp + 0x40
0x01f23e10 call fcn.01f29eee
0x01f23e15 test eax, eax
0x01f23e17 push 0
0x01f23e19 push dword [edi + 0x80
0x01f23e1f push dword [ebp + 0x98
0x01f23e25 cmp dh, 0x8a ; 1
0x01f23e28 call fcn.01f29eee
0x01f23e2d test bh, 3 ; 3
0x01f23e30 cmp bl, bl
0x01f23e32 add dword [ebp + 0x11c
0x01f23e39 cmp ah, dh
0x01f23e3b push 0x0

```

Indicators of Compromise:-

- **Host Based IOC**
 - File hash :-
759c87f02d5850ee317454dc8242067d042a320f653946c9162b645c0ae68ba6

Conclusion:- Guloder is a highly evasive windows malware downloader. As seen in the analysis, it uses multiple checks at various points for VM and debugger presence. The code also contains useless instructions all along. These features make malware downloaders like Guloder highly attractive for threat actor's and malware developer's. Instead of directly executing the malware on the victim's machine, guloder provide a way to check the execution environment to see if inside a vm or debugger and If needed download the actual payload which can be another malware.