The evolution of GuLoader

vmray.com/cyber-security-blog/malware-analysis-spotlight-guloader

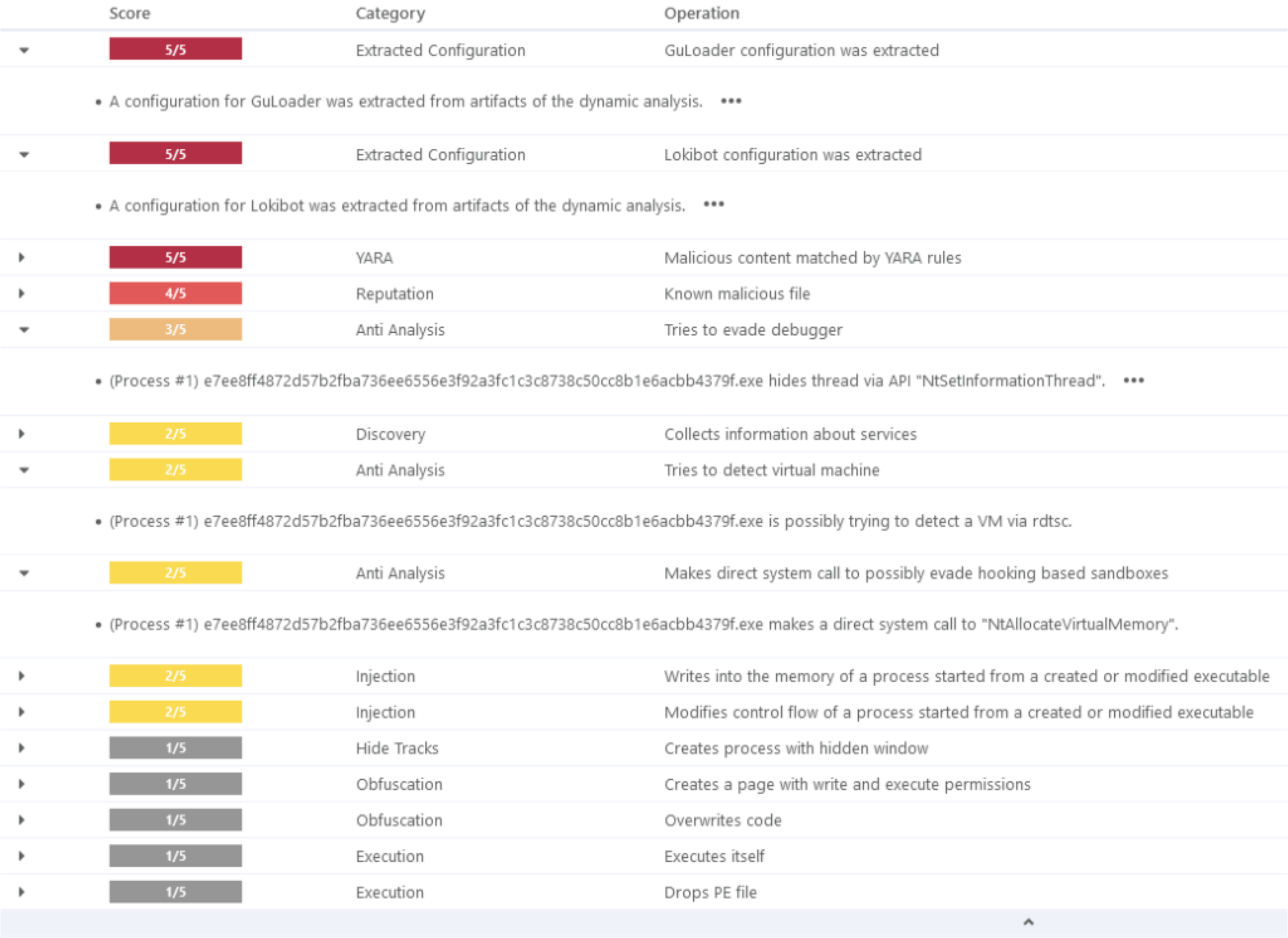
The evolution of GuLoader
MALWARE ANALYSIS SPOTLIGHT FROM VMRAY LABS

Table of Contents

Introduction

In this Spotlight, we take another look at GuLoader. The malware family is active since at least 2020. It gained some attention because of its evasion techniques and abusing legitimate and popular cloud services to host its malicious payloads. The downloader is commonly used to deliver other malware families such as FormBook, XLoader, and Lokibot. After we took a closer look at GuLoader’s evasion techniques in a Threat Bulletin, we observed some additional behavior later that year.

Recently, we collected samples that are different from the samples we have seen before. The file that executes GuLoader’s shellcode has changed, and the functionality of GuLoader has been extended compared to our last Spotlight. The sample in discussion leads to the execution of Lokibot as indicated by the extracted configurations in Figure 1.



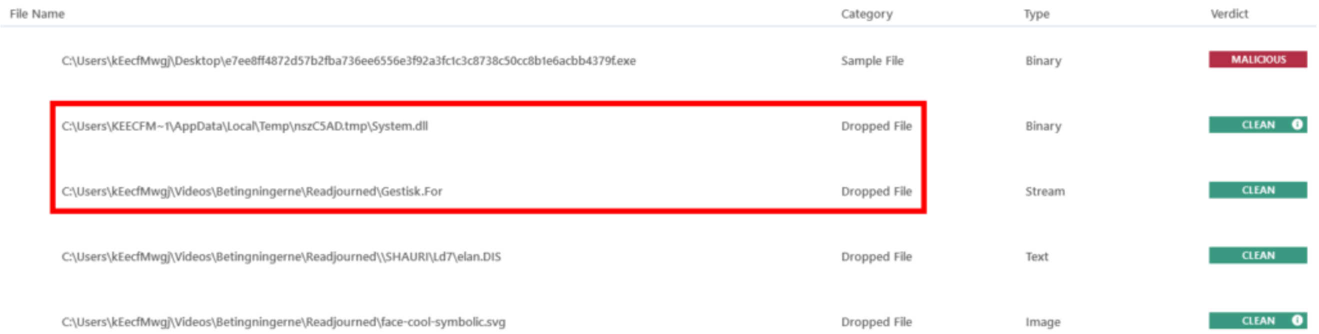
Score	Category	Operation
5/5	Extracted Configuration	GuLoader configuration was extracted
	• A configuration for GuLoader was extracted from artifacts of the dynamic analysis. •••	
5/5	Extracted Configuration	Lokibot configuration was extracted
	• A configuration for Lokibot was extracted from artifacts of the dynamic analysis. •••	
5/5	YARA	Malicious content matched by YARA rules
4/5	Reputation	Known malicious file
3/5	Anti Analysis	Tries to evade debugger
	• (Process #1) e7ee8ff4872d57b2fba736ee6556e3f92a3fc1c3c8738c50cc8b1e6acbb4379f.exe hides thread via API "NtSetInformationThread". •••	
2/5	Discovery	Collects information about services
2/5	Anti Analysis	Tries to detect virtual machine
	• (Process #1) e7ee8ff4872d57b2fba736ee6556e3f92a3fc1c3c8738c50cc8b1e6acbb4379f.exe is possibly trying to detect a VM via rdtsc.	
2/5	Anti Analysis	Makes direct system call to possibly evade hooking based sandboxes
	• (Process #1) e7ee8ff4872d57b2fba736ee6556e3f92a3fc1c3c8738c50cc8b1e6acbb4379f.exe makes a direct system call to "NtAllocateVirtualMemory".	
2/5	Injection	Writes into the memory of a process started from a created or modified executable
2/5	Injection	Modifies control flow of a process started from a created or modified executable
1/5	Hide Tracks	Creates process with hidden window
1/5	Obfuscation	Creates a page with write and execute permissions
1/5	Obfuscation	Overwrites code
1/5	Execution	Executes itself
1/5	Execution	Drops PE file

Figure 1: VMRay Analyzer - VTI highlighting GuLoader's behavior and extracted configurations.

GuLoader’s Delivery

The main functionality of GuLoader is implemented as shellcode, and typically an executable takes care of loading the shellcode into memory and transferring the execution flow to it. So far this executable was written in VB6. However, the executable in this analysis is a signed NSIS installer that leads to the execution of GuLoader.

During the installation process, the installer extracts multiple files to the hard disk including a DLL (Dynamic Link Library) named “System.dll”, and a file named “Gestisk.For” (Figure 2.).



File Name	Category	Type	Verdict
C:\Users\kEecfMwgj\Desktop\e7ee8ff4872d57b2fba736ee6556e3f92a3fc1c3c8738c50cc8b1e6acbb4379f.exe	Sample File	Binary	MALICIOUS
C:\Users\KEECFM~1\AppData\Local\Temp\nszC5AD.tmp\System.dll	Dropped File	Binary	CLEAN
C:\Users\kEecfMwgj\Videos\Betingningerne\Readjourned\Gestisk.For	Dropped File	Stream	CLEAN
C:\Users\kEecfMwgj\Videos\Betingningerne\Readjourned\\SHAURI\Ld7\elan.DIS	Dropped File	Text	CLEAN
C:\Users\kEecfMwgj\Videos\Betingningerne\Readjourned\face-cool-symbolic.svg	Dropped File	Image	CLEAN

Figure 2: VMRay Analyzer - Dropped files

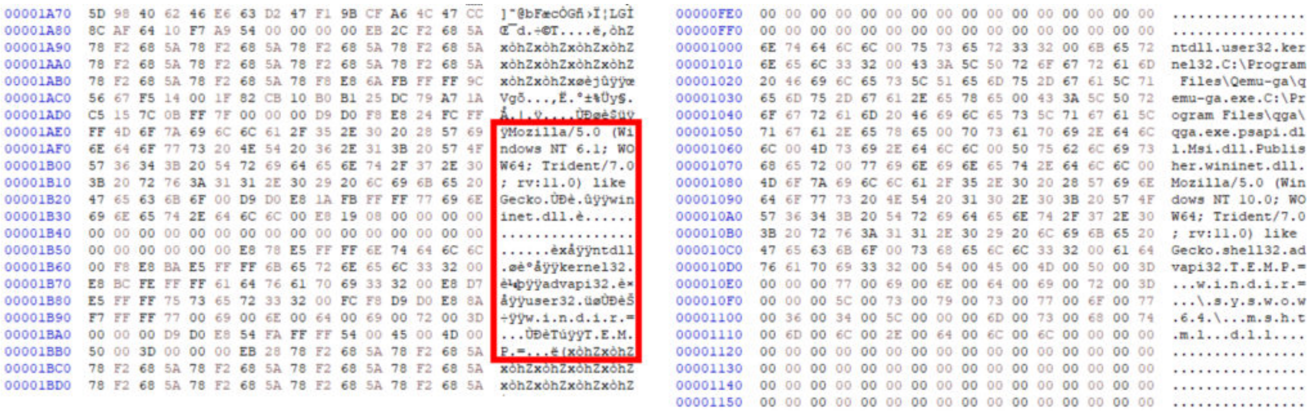
While the name for the DLL seems to be consistent across similar samples, the name of the second file can vary. After writing "System.dll" to the hard disk, it is loaded by the installer and used to call WinAPI functions to allocate memory where the shellcode will end up alter on.

Previous samples written in VB6 called the WinAPI functions directly instead of using a separate DLL.

GuLoader's Evolution

At first glance, we can see the typical behavior of GuLoader. It tries to detect an analysis environment and if none was found it injects the shellcode into another process instance of the executable.

Next, the second instance downloads and executes the payload from the well-known cloud service Google Drive. When comparing the memory dump of the shellcode with memory dumps from older samples, we can see that GuLoader stopped storing the strings in plaintext. Instead, they are decrypted at runtime and stored in a separate memory region (Figure 3.).



```
00001A70  5D 98 40 62 46 E6 63 D2 47 F1 9B CF A6 4C 47 CC  ]˜@bFæcÒGñ›Ï¦LGÌ
00001A80  8C AF 64 10 F7 A9 54 00 00 00 00 EB 2C F2 68 5A  Œ¯d.÷©T....ë,òhZ
00001A90  78 F2 68 5A 78 F2 68 5A 78 F2 68 5A 78 F2 68 5A  xòhZxòhZxòhZxòhZ
00001AA0  78 F2 68 5A 78 F2 68 5A 78 F2 68 5A 78 F2 68 5A  xòhZxòhZxòhZxòhZ
00001AB0  78 F2 68 5A 78 F2 68 5A 78 F8 E8 6A FB FF FF 9C  xòhZxòhZxøèjûÿÿœ
00001AC0  56 67 F5 14 00 1F 82 CB 10 B0 B1 25 DC 79 A7 1A  Vgõ...,Ë.°±%Üy§.
00001AD0  C5 15 7C 0B FF 7F 00 00 00 D9 D0 F8 E8 24 FC FF  Å.|.ÿ....ÙÐøè$üÿ
00001AE0  FF 4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57 69  ÿMozilla/5.0 (Wi
00001AF0  6E 64 6F 77 73 20 4E 54 20 36 2E 31 3B 20 57 4F  ndows NT 6.1; WO
00001B00  57 36 34 3B 20 54 72 69 64 65 6E 74 2F 37 2E 30  W64; Trident/7.0
00001B10  3B 20 72 76 3A 31 31 2E 30 29 20 6C 69 6B 65 20  ; rv:11.0) like 
00001B20  47 65 63 6B 6F 00 D9 D0 E8 1A FB FF FF 77 69 6E  Gecko.ÙÐè.ûÿÿwin
00001B30  69 6E 65 74 2E 64 6C 6C 00 E8 19 08 00 00 00 00  inet.dll.è......
00001B40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001B50  00 00 00 00 00 00 E8 78 E5 FF FF 6E 74 64 6C 6C  ......èxåÿÿntdll
00001B60  00 F8 E8 BA E5 FF FF 6B 65 72 6E 65 6C 33 32 00  .øèºåÿÿkernel32.
00001B70  E8 BC FE FF FF 61 64 76 61 70 69 33 32 00 E8 D7  è¼þÿÿadvapi32.è×
00001B80  E5 FF FF 75 73 65 72 33 32 00 FC F8 D9 D0 E8 8A  åÿÿuser32.üøÙÐèŠ
00001B90  F7 FF FF 77 00 69 00 6E 00 64 00 69 00 72 00 3D  ÷ÿÿw.i.n.d.i.r.=
00001BA0  00 00 00 D9 D0 E8 54 FA FF FF 54 00 45 00 4D 00  ...ÙÐèTúÿÿT.E.M.
00001BB0  50 00 3D 00 00 00 EB 28 78 F2 68 5A 78 F2 68 5A  P.=...ë(xòhZxòhZ
00001BC0  78 F2 68 5A 78 F2 68 5A 78 F2 68 5A 78 F2 68 5A  xòhZxòhZxòhZxòhZ
00001BD0  78 F2 68 5A 78 F2 68 5A 78 F2 68 5A 78 F2 68 5A  xòhZxòhZxòhZxòhZ
```

```
00000FE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001000  6E 74 64 6C 6C 00 75 73 65 72 33 32 00 6B 65 72  ntdll.user32.ker
00001010  6E 65 6C 33 32 00 43 3A 5C 50 72 6F 67 72 61 6D  nel32.C:\Program
00001020  20 46 69 6C 65 73 5C 51 65 6D 75 2D 67 61 5C 71   Files\Qemu-ga\q
00001030  65 6D 75 2D 67 61 2E 65 78 65 00 43 3A 5C 50 72  emu-ga.exe.C:\Pr
00001040  6F 67 72 61 6D 20 46 69 6C 65 73 5C 71 67 61 5C  ogram Files\qga\
00001050  71 67 61 2E 65 78 65 00 70 73 61 70 69 2E 64 6C  qga.exe.psapi.dl
00001060  6C 00 4D 73 69 2E 64 6C 6C 00 50 75 62 6C 69 73  l.Msi.dll.Publis
00001070  68 65 72 00 77 69 6E 69 6E 65 74 2E 64 6C 6C 00  her.wininet.dll.
00001080  4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57 69 6E  Mozilla/5.0 (Win
00001090  64 6F 77 73 20 4E 54 20 31 30 2E 30 3B 20 57 4F  dows NT 10.0; WO
000010A0  57 36 34 3B 20 54 72 69 64 65 6E 74 2F 37 2E 30  W64; Trident/7.0
000010B0  3B 20 72 76 3A 31 31 2E 30 29 20 6C 69 6B 65 20  ; rv:11.0) like 
000010C0  47 65 63 6B 6F 00 73 68 65 6C 6C 33 32 00 61 64  Gecko.shell32.ad
000010D0  76 61 70 69 33 32 00 54 00 45 00 4D 00 50 00 3D  vapi32.T.E.M.P.=
000010E0  00 00 00 77 00 69 00 6E 00 64 00 69 00 72 00 3D  ...w.i.n.d.i.r.=
000010F0  00 00 00 5C 00 73 00 79 00 73 00 77 00 6F 00 77  ...\.s.y.s.w.o.w
00001100  00 36 00 34 00 5C 00 00 00 6D 00 73 00 68 00 74  .6.4.\...m.s.h.t
00001110  00 6D 00 6C 00 2E 00 64 00 6C 00 6C 00 00 00 00  .m.l...d.l.l....
00001120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 3: Encrypted strings embedded in shellcode (left), and decrypted strings stored in a separate memory region (right).

VMRay Analyzer uses special triggers that allow obtaining the region which contains the decrypted strings.

Moving on to the observed function calls, we can see that the sample utilizes additional WinAPI functions compared to previous ones. Figure 4. lists additional function calls that we discuss next.

RtlAddVectoredExceptionHandler

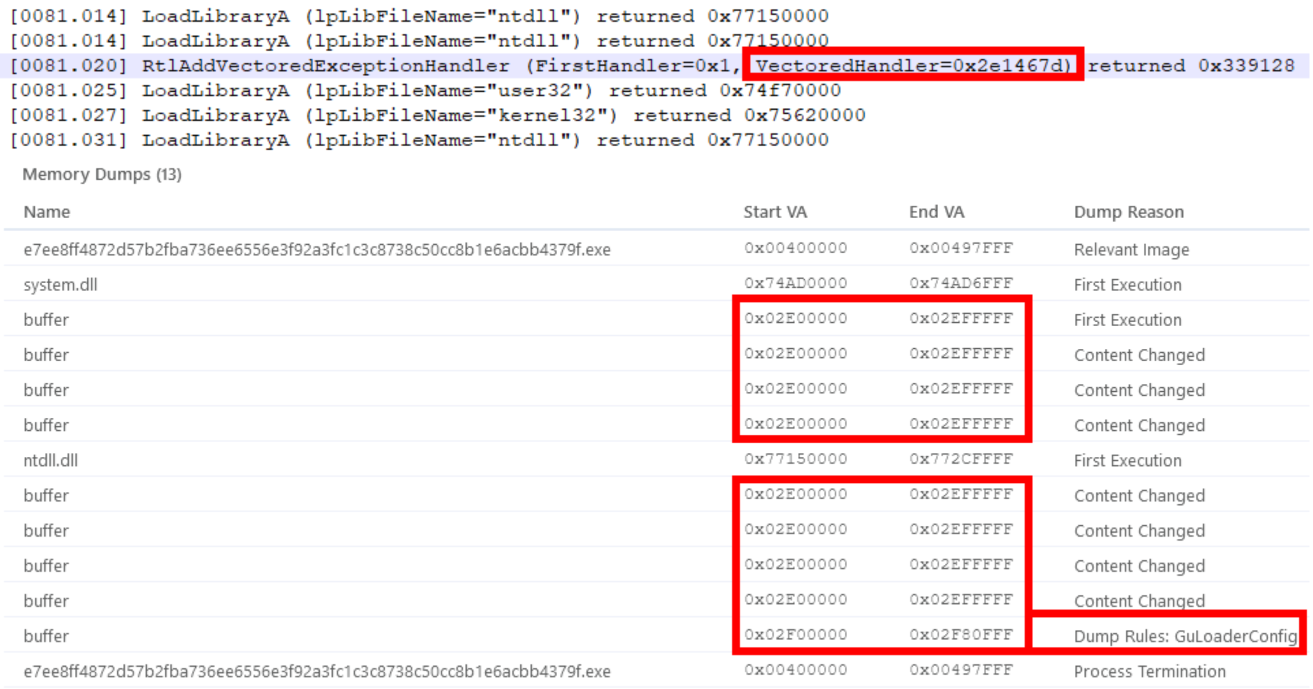
EnumDeviceDrivers
GetDeviceDriverBaseNameA

MsiEnumProductsA
MsiGetProductInfoA

OpenSCManagerA
EnumServicesStatusA
Figure 4: List of additional WinAPI functions observed in newer samples.

While we have seen calls to functions related to enumerating products and services in previous samples, the registration of a new exception handler and the examination of device drivers have been added recently. This leads to the assumption that GuLoader is still under active development.

Given the function log, we can see that the address of the exception handler is part of the shellcode (Figure 5.).



```
[0081.014] LoadLibraryA (lpLibFileName="ntdll") returned 0x77150000
[0081.014] LoadLibraryA (lpLibFileName="ntdll") returned 0x77150000
[0081.020] RtlAddVectoredExceptionHandler (FirstHandler=0x1, VectoredHandler=0x2e1467d) returned 0x339128
[0081.025] LoadLibraryA (lpLibFileName="user32") returned 0x74f70000
[0081.027] LoadLibraryA (lpLibFileName="kernel32") returned 0x75620000
[0081.031] LoadLibraryA (lpLibFileName="ntdll") returned 0x77150000
```

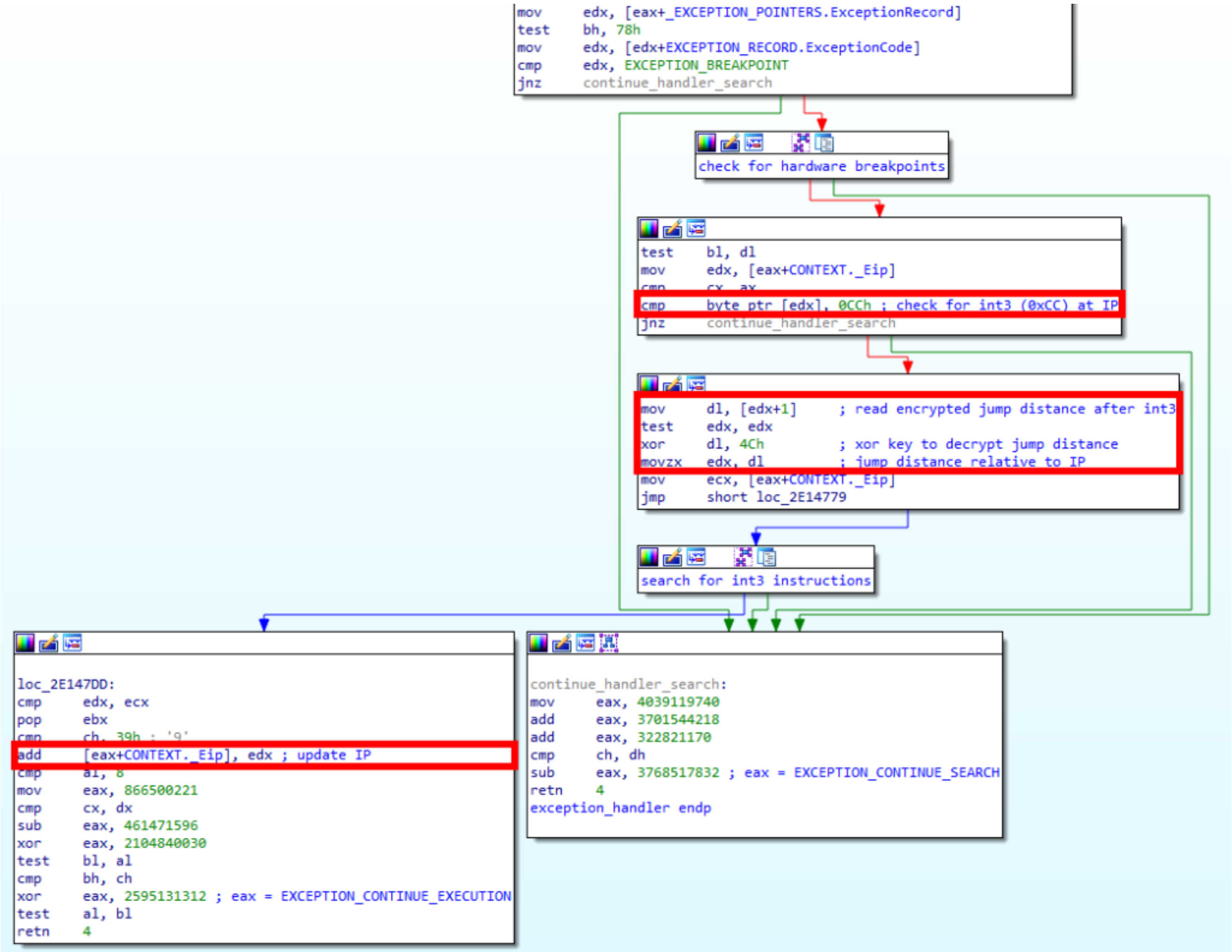
Memory Dumps (13)

Name	Start VA	End VA	Dump Reason
e7ee8ff4872d57b2fba736ee6556e3f92a3fc1c3c8738c50cc8b1e6acbb4379f.exe	0x00400000	0x00497FFF	Relevant Image
system.dll	0x74AD0000	0x74AD6FFF	First Execution
buffer	0x02E00000	0x02EFFFFF	First Execution
buffer	0x02E00000	0x02EFFFFF	Content Changed
buffer	0x02E00000	0x02EFFFFF	Content Changed
buffer	0x02E00000	0x02EFFFFF	Content Changed
ntdll.dll	0x77150000	0x772CFFFF	First Execution
buffer	0x02E00000	0x02EFFFFF	Content Changed
buffer	0x02E00000	0x02EFFFFF	Content Changed
buffer	0x02E00000	0x02EFFFFF	Content Changed
buffer	0x02E00000	0x02EFFFFF	Content Changed
buffer	0x02F00000	0x02F80FFF	Dump Rules: GuLoaderConfig
e7ee8ff4872d57b2fba736ee6556e3f92a3fc1c3c8738c50cc8b1e6acbb4379f.exe	0x00400000	0x00497FFF	Process Termination

Figure 5: VMRay Analyzer - Exception handler registration

This exception handler first checks if the exception was raised because of a software breakpoint. Next, the function inspects the CPU registers to detect the presence of hardware breakpoints. In case no breakpoint is set, the handler continues to change the instruction pointer. The new value depends on the current instruction pointer and the byte followed after the int3 instruction that triggered the exception handler (Figure 6). If a hardware breakpoint is set, the handler doesn't change the instruction pointer, subsequently executing invalid instructions.

Additionally, the function checks for int3 instructions between the current and the new instruction pointer value.



```
mov     edx, [eax+_EXCEPTION_POINTERS.ExceptionRecord]
test    bh, 78h
mov     edx, [edx+EXCEPTION_RECORD.ExceptionCode]
cmp     edx, EXCEPTION_BREAKPOINT
jnz     continue_handler_search
```

```
check for hardware breakpoints
```

```
test    bl, dl
mov     edx, [eax+CONTEXT._Eip]
cmp     cx, ax
cmp     byte ptr [edx], 0CCh ; check for int3 (0xCC) at IP
jnz     continue_handler_search
```

```
mov     dl, [edx+1]      ; read encrypted jump distance after int3
test    edx, edx
xor     dl, 4Ch          ; xor key to decrypt jump distance
movzx   edx, dl          ; jump distance relative to IP
mov     ecx, [eax+CONTEXT._Eip]
jmp     short loc_2E14779
```

```
search for int3 instructions
```

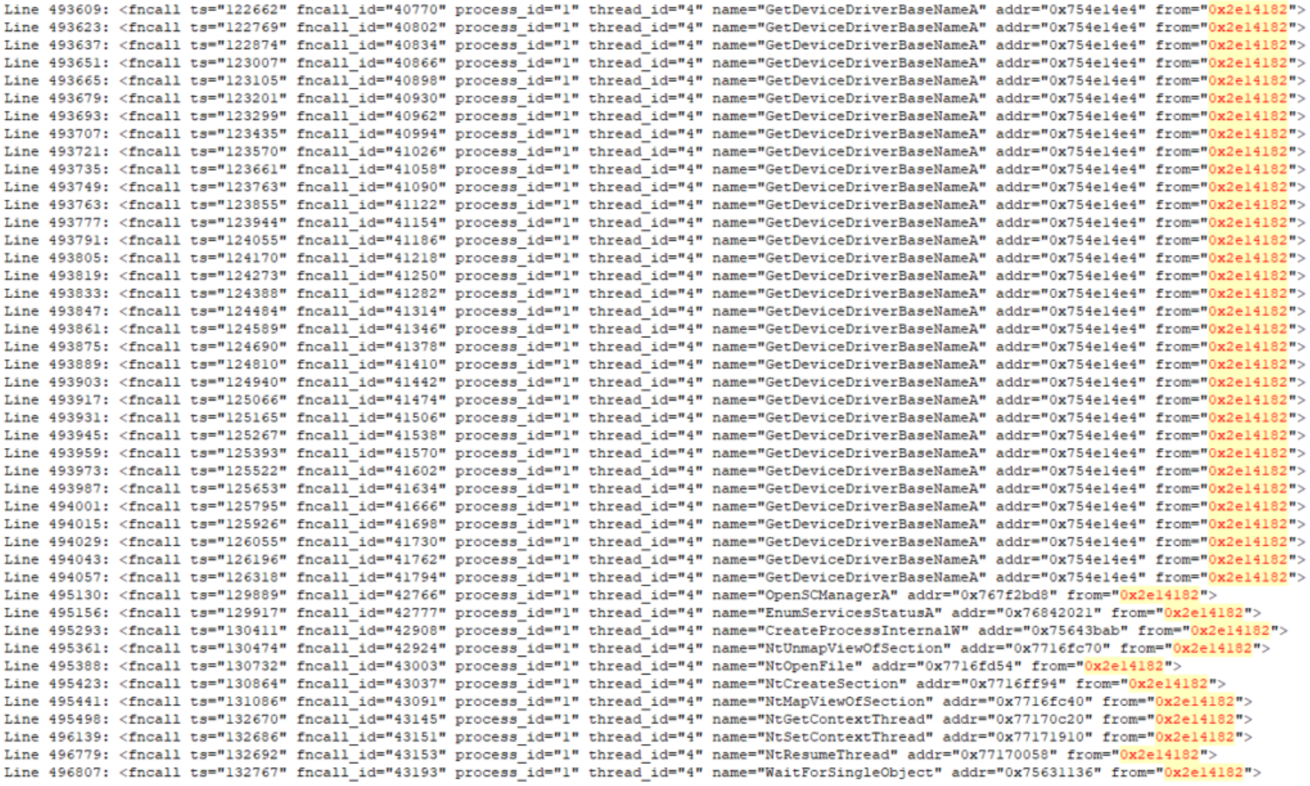
```
loc_2E147DD:
cmp     edx, ecx
pop     ebx
cmp     ch, 39h ; '9'
add     [eax+CONTEXT._Eip], edx ; update IP
cmp     al, 8
mov     eax, 866500221
cmp     cx, dx
sub     eax, 461471596
xor     eax, 2104840030
test    bl, al
cmp     bh, ch
xor     eax, 2595131312 ; eax = EXCEPTION_CONTINUE_EXECUTION
test    al, bl
retn    4
```

```
continue_handler_search:
mov     eax, 4039119740
add     eax, 3701544218
add     eax, 322821170
cmp     ch, dh
sub     eax, 3768517832 ; eax = EXCEPTION_CONTINUE_SEARCH
retn    4
exception_handler endp
```

Figure 6: Exception handler snippet that modifies the instruction pointer.

By registering the exception handler, GuLoader uses int3 instructions as relative jumps. Because debuggers like WinDbg and x64dbg use int3 instructions for software breakpoints, this approach interferes with debugging if the debugger handles these exceptions first.

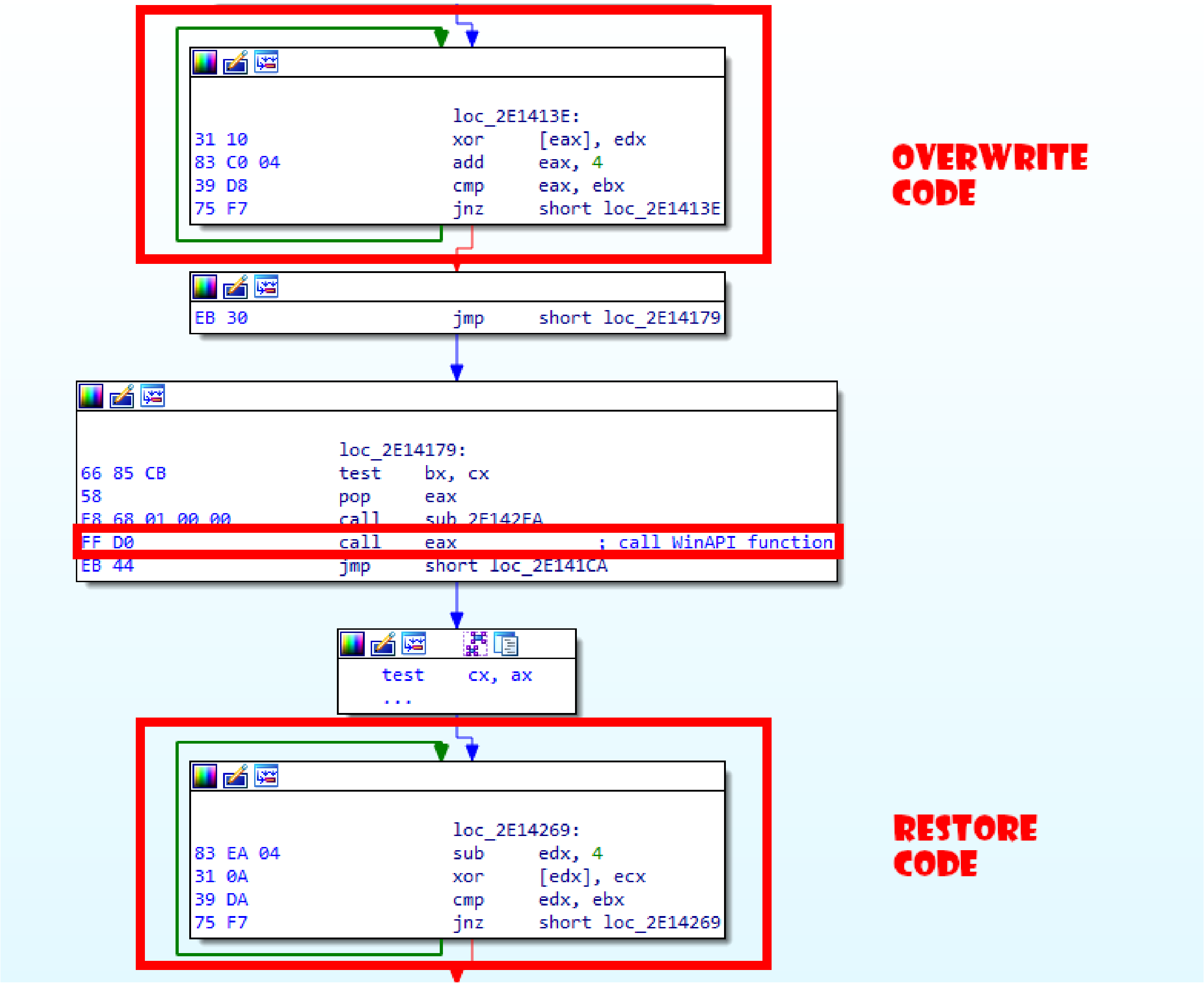
A deeper look at the function log reveals that multiple WinAPI functions are called from the same address within the shellcode (Figure 7.). This is an indicator that some kind of wrapper function takes care of calling the WinAPI functions.



```
Line 493609: <fncall ts="122662" fncall_id="40770" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493623: <fncall ts="122769" fncall_id="40802" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493637: <fncall ts="122874" fncall_id="40834" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493651: <fncall ts="123007" fncall_id="40866" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493665: <fncall ts="123105" fncall_id="40898" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493679: <fncall ts="123201" fncall_id="40930" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493693: <fncall ts="123299" fncall_id="40962" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493707: <fncall ts="123435" fncall_id="40994" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493721: <fncall ts="123570" fncall_id="41026" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493735: <fncall ts="123661" fncall_id="41058" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493749: <fncall ts="123763" fncall_id="41090" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493763: <fncall ts="123855" fncall_id="41122" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493777: <fncall ts="123944" fncall_id="41154" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493791: <fncall ts="124055" fncall_id="41186" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493805: <fncall ts="124170" fncall_id="41218" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493819: <fncall ts="124273" fncall_id="41250" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493833: <fncall ts="124388" fncall_id="41282" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493847: <fncall ts="124484" fncall_id="41314" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493861: <fncall ts="124589" fncall_id="41346" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493875: <fncall ts="124690" fncall_id="41378" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493889: <fncall ts="124810" fncall_id="41410" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493903: <fncall ts="124940" fncall_id="41442" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493917: <fncall ts="125066" fncall_id="41474" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493931: <fncall ts="125165" fncall_id="41506" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493945: <fncall ts="125267" fncall_id="41538" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493959: <fncall ts="125393" fncall_id="41570" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493973: <fncall ts="125522" fncall_id="41602" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 493987: <fncall ts="125653" fncall_id="41634" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 494001: <fncall ts="125795" fncall_id="41666" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 494015: <fncall ts="125926" fncall_id="41698" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 494029: <fncall ts="126055" fncall_id="41730" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 494043: <fncall ts="126196" fncall_id="41762" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 494057: <fncall ts="126318" fncall_id="41794" process_id="1" thread_id="4" name="GetDeviceDriverBaseNameA" addr="0x754e14e4" from="0x2e14182">
Line 495130: <fncall ts="129889" fncall_id="42766" process_id="1" thread_id="4" name="OpenSCManagerA" addr="0x767f2bd8" from="0x2e14182">
Line 495156: <fncall ts="129917" fncall_id="42777" process_id="1" thread_id="4" name="EnumServicesStatusA" addr="0x76842021" from="0x2e14182">
Line 495293: <fncall ts="130411" fncall_id="42908" process_id="1" thread_id="4" name="CreateProcessInternalW" addr="0x75643bab" from="0x2e14182">
Line 495361: <fncall ts="130474" fncall_id="42924" process_id="1" thread_id="4" name="NtUnmapViewOfSection" addr="0x7716fc70" from="0x2e14182">
Line 495388: <fncall ts="130732" fncall_id="43003" process_id="1" thread_id="4" name="NtOpenFile" addr="0x7716fd54" from="0x2e14182">
Line 495423: <fncall ts="130864" fncall_id="43037" process_id="1" thread_id="4" name="NtCreateSection" addr="0x7716ff94" from="0x2e14182">
Line 495441: <fncall ts="131086" fncall_id="43091" process_id="1" thread_id="4" name="NtMapViewOfSection" addr="0x7716fc40" from="0x2e14182">
Line 495498: <fncall ts="132670" fncall_id="43145" process_id="1" thread_id="4" name="NtGetContextThread" addr="0x77170c20" from="0x2e14182">
Line 496139: <fncall ts="132686" fncall_id="43151" process_id="1" thread_id="4" name="NtSetContextThread" addr="0x77171910" from="0x2e14182">
Line 496779: <fncall ts="132692" fncall_id="43153" process_id="1" thread_id="4" name="NtResumeThread" addr="0x77170058" from="0x2e14182">
Line 496807: <fncall ts="132767" fncall_id="43193" process_id="1" thread_id="4" name="WaitForSingleObject" addr="0x75631136" from="0x2e14182">
```

Figure 7: VMRay Analyzer - Excerpt from flog.xml revealing the same from address is being used multiple times.

In this example, GuLoader uses such a function to partially overwrite its code before calling the actual WinAPI function. Figure 8. shows the part of the wrapper function that overwrites the code by xoring it with the return address before and after the call instruction.



```
                          loc_2E1413E:
31 10                     xor     [eax], edx
83 C0 04                  add     eax, 4
39 D8                     cmp     eax, ebx
75 F7                     jnz     short loc_2E1413E

EB 30                     jmp     short loc_2E14179

                          loc_2E14179:
66 85 CB                  test    bx, cx
58                        pop     eax
E8 68 01 00 00            call    sub_2E142EA
FF D0                     call    eax                 ; call WinAPI function
EB 44                     jmp     short loc_2E141CA

        test    cx, ax
        ...

                          loc_2E14269:
83 EA 04                  sub     edx, 4
31 0A                     xor     [edx], ecx
39 DA                     cmp     edx, ebx
75 F7                     jnz     short loc_2E14269
```

Figure 8: Partially overwriting code before WinAPI function calls.

By overwriting code before the calls, GuLoader **avoids being extracted correctly by analysis tools that use WinAPI functions** as memory dump trigger.

Looking at the list of called functions, we can see that GuLoader gathers information about the

- name of installed drivers EnumDeviceDrivers and GetDeviceDriverBaseNameA)
- publisher of installed products (MsiEnumProductsA and MsiGetProductInfoA)
- services in the SERVICES_ACTIVE_DATABASE

The resulting strings are then hashed using a customized djb2 algorithm and compared against a block list of pre-computed values of analysis environment artifacts.

Device names:

0x0A4F1B4F0
0x0D277D8C6

```
0x06E5A1CF8
0x0966FE6F7
0x0EC7C85F9
```

Product publisher:

```
0x07630654D
0x0A80331E9
0x0F8727F49
0x060FAFADD
```

Services:

```
0x0C749257D
0x0CC359518
0x0C55733D2
0x0A0F0EF16
0x0BA252FC4
0x02DC0E42A
0x077C8F76A
```

Figure 9: Blocklist of pre-computed values of analysis environment artifacts

If the calculated value is present in the block list, GuLoader stops its execution and therefore evades the analysis.

This technique was used earlier with the original djb2 algorithm. In this particular sample, the djb2 algorithm is customized in a way that the hash is xored with the key 0x0C93EB2D8 in each iteration (Figure 9.)

```python
def djb2_custom(s: bytes) -> int:
    hash = 5381
    for x in s:
        hash = ((hash << 5) + hash) + x
        hash = (hash ^ 0x0C93EB2D8) & 0xFFFFFFFF
    return hash
```

Figure 10: Customized djb2 algorithm in Python

In general, values of the block list are indicators analysts can take advantage of for detection and identification as long as the algorithm remains the same across samples. GuLoader prevents this by slightly changing the algorithm.

Finally, GuLoader creates another process of the installer, injects code, and delivers the payload. In this case, the payload is Lokibot and hosted on Google Drive.

VMRay Analyzer extracts the malware configuration for both malware families, which eases the detection and identification of infected systems.

Extracted Payload URLs

In addition to Google Drive being abused to host the malicious payload, we have seen other services in our extracted configurations.

Figure 10. shows the distribution of hostnames. While Google Drive remains the most common one, other cloud services like Microsoft OneDrive are used a well.

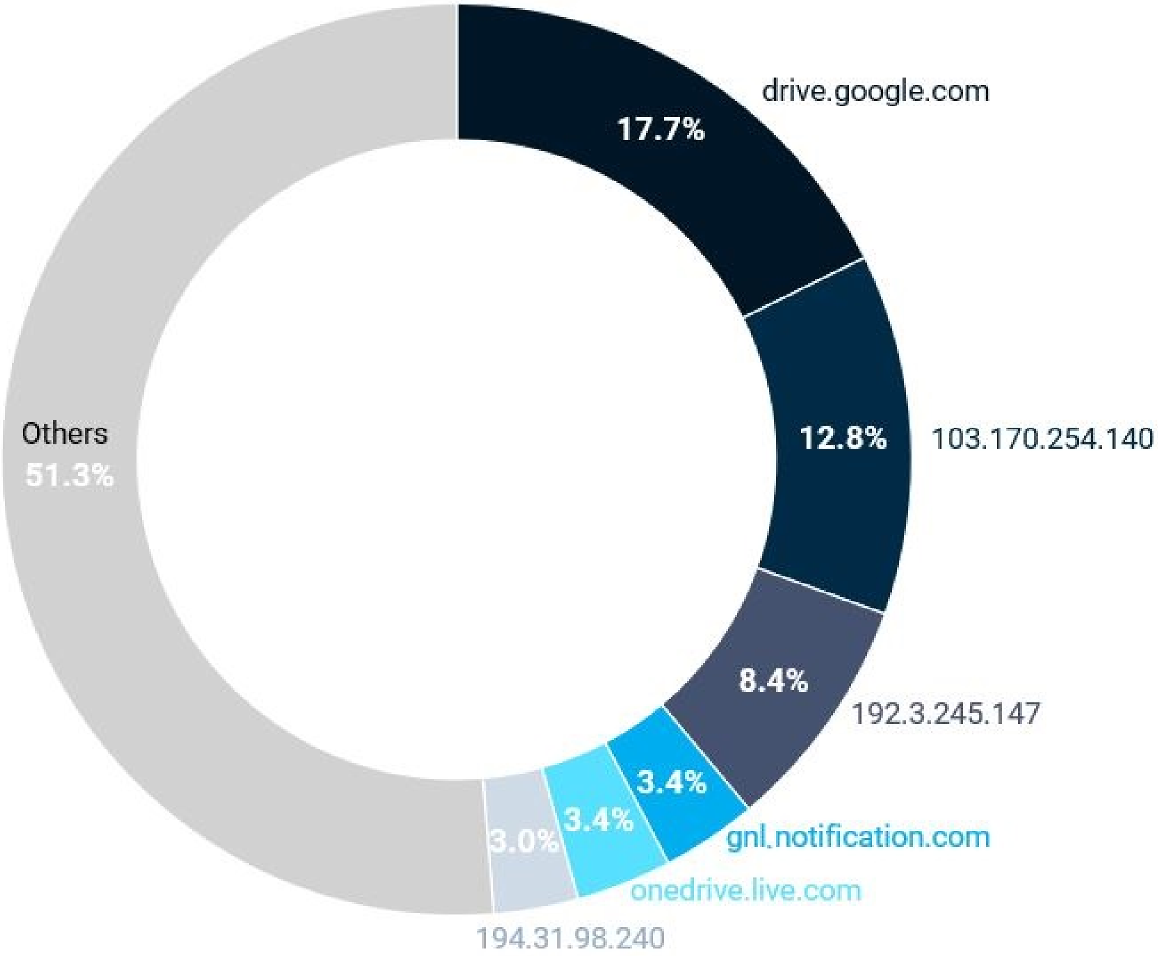


Figure 11: Distribution of host names

Conclusion

In this post, we took another look at GuLoader with a focus on behavioral differences compared to past samples. We have seen that not only the executable, which leads to GuLoader’s shellcode has been changed but also its functionality has been further extended.

While GuLoader utilizes new techniques to search for artifacts revealing an analysis environment, some of the existing logic changed to further thwart detection and analysis attempts. Given VMRay Analyzer’s unique monitoring approach, GuLoader can’t detect the presence of the sandbox and reveal its malicious behavior leading to the delivery of Lokibot. The extracted malware configuration for both families allows analysts and incident responders to quickly take actions to prevent the infection and identify already compromised machines.

IOCs

Initial Sample:

e7ee8ff4872d57b2fba736ee6556e3f92a3fc1c3c8738c50cc8b1e6acbb4379f

GuLoader Payload URL:

hxxps://drive[.]google[.]com/uc?export=download&id=1SrbfkJ9_Bx7Q9qhzb5JeLy5TlBRjWwjF

Lokibot C&Cs:

alphastand[.]trade/alien/fre.php

alphastand[.]top/alien/fre.php

alphastand[.]win/alien/fre.php

kbfvzoboss[.]bid/alien/fre.php

hxxp://198[.]187[.]30[.]47/p.php?id=67243588715181780

Pascal Brackmann
Pascal is a Threat Researcher at VMRay Labs. His recent projects cover in-depth analysis of emerging and evolving malware.