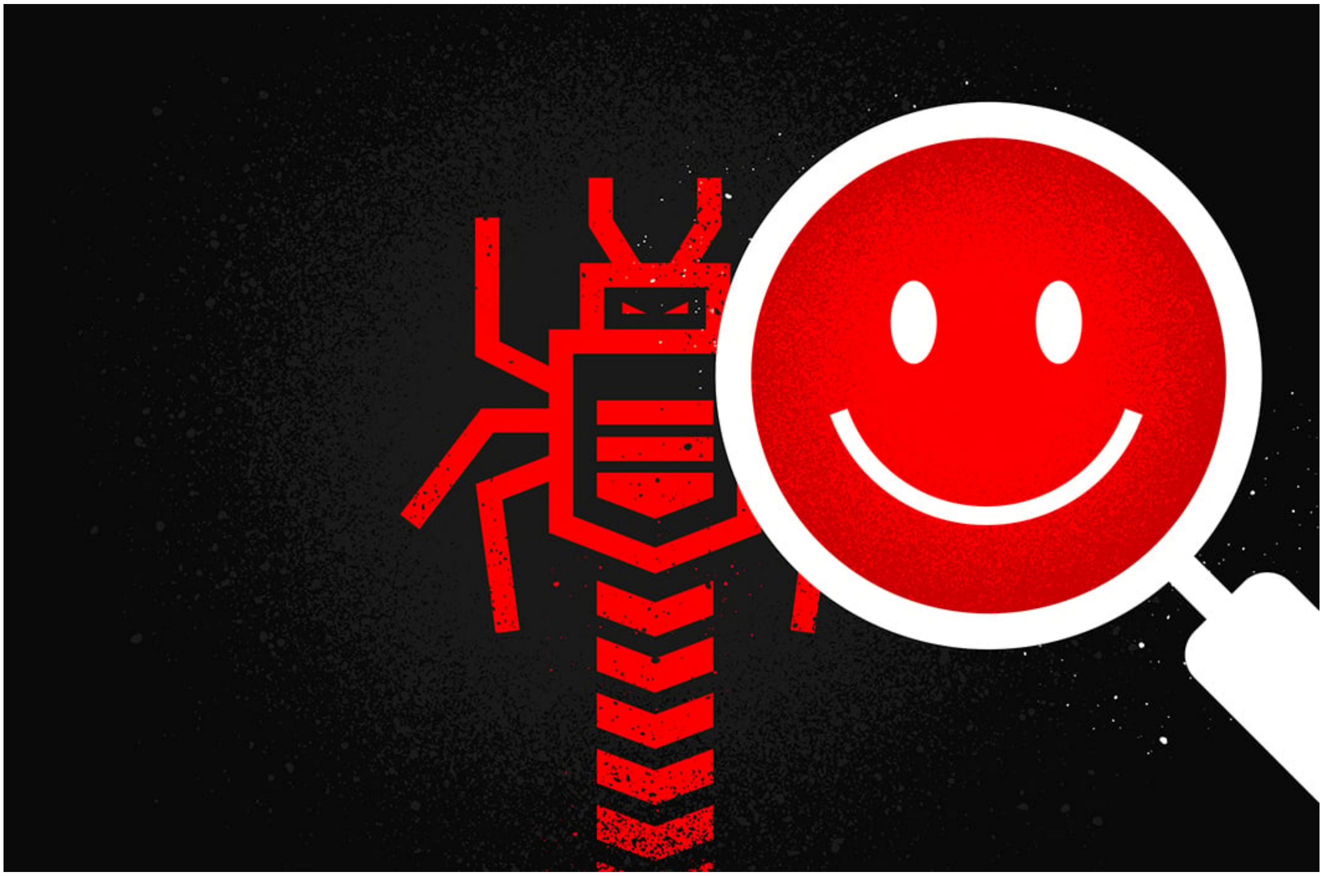
GuLoader Dissection Reveals New Anti-Analysis

crowdstrike.com/blog/guloader-dissection-reveals-new-anti-analysis-techniques-and-code-injection-redundancy/

Sarang Sonawane - Donato Onofri

December 19, 2022



- GuLoader is an advanced malware downloader that uses a polymorphic shellcode loader to dodge traditional security solutions
- CrowdStrike researchers expose complete GuLoader behavior by mapping all embedded DJB2 hash values for every API used by the malware
- New shellcode anti-analysis technique attempts to thwart researchers and hostile environments by scanning entire process memory for any virtual machine (VM)-related strings
- New redundant code injection mechanism means to ensure code execution by using inline assembly to bypass user mode hooks from security solutions

CrowdStrike analyzes malware to augment the behavior and machine learning-based detection and protection capabilities built into the CrowdStrike Falcon® platform to deliver automated, world-class protection to customers.

GuLoader has been known to employ a significant number of anti-analysis techniques, making detection and protection challenging for other security solutions.

In dissecting GuLoader's shellcode, CrowdStrike revealed a new anti-analysis technique meant to detect if the malware is running in a hostile environment by scanning the entire process memory for any Virtual Machine (VM)-related strings. To bypass GuLoader's anti-debugging evasion mechanisms, we found and described two new working methods for patching debugger instructions meant to detect the presence of debugging tools used by researchers for analysis.

This analysis includes what we believe is the first-ever mapping of all remaining DJB2 hash values for every API used by the GuLoader malware, revealing the first-ever complete view into the malware's behavior and how it interacts with the victim's machine.

See for yourself how the industry-leading CrowdStrike Falcon platform protects against modern threats like GuLoader. Start your 15-day free trial today.

The Evolution of GuLoader

GuLoader was first spotted in 2019 as a file downloader that was used to distribute remote access trojans (RATs) such as AgentTesla, FormBook, Nanocore, NETWIRE and the Parallax RAT.

These early versions of GuLoader were distributed via spam email campaigns containing archived attachments containing the malware in executable form. In 2020, CrowdStrike published a detailed analysis of GuLoader in which a significant number of DJB2 hash values were mapped, revealing some of the APIs abused by the malware.

Recent variants started using an updated delivery mechanism where the payload is delivered via a Visual Basic Script (VBS) file. GuLoader also started employing advanced anti-analysis techniques to evade detection, such as anti-debug, anti-sandbox, anti-VM and anti-detection to make analysis difficult.

By analyzing the new GuLoader samples, we're able to reveal and understand every anti-analysis and evasion technique being employed and paint a complete picture of the malware's behavior.

GuLoader's Multistage Deployment

The recent GuLoader sample exhibits a multistage deployment:

- **The first stage** involves using a VBS dropper file to drop a second-stage packed payload into a registry key. It then uses a PowerShell script to execute and unpack the second stage payload from the registry key within memory.
- **The second stage** payload performs all anti-analysis routines (described below), creates a Windows process (e.g., an ieinstal.exe) and injects the same shellcode into the new process.

- **The third stage** reimplements all the anti-analysis techniques, downloads the final payload from a remote server and executes it on the victim's machine.

Existing public research on GuLoader's multistage deployment has extensively covered a wide range of anti-evasion techniques and behaviors. We used this as a starting point to further our analysis.

VBScript

The VBScript contains two main arrays. One of them is the shellcode present in hex format that will be injected into memory and the second is a Base64-encrypted PowerShell script.

For persistence, this shellcode is then added to the Registry Key (HKEY_CURRENT_USER\SOFTWARE\TYMPANIESI) by the VBScript.

Another variable "MEDITABU" with Base64 content is being merged and after decoding it forms a PowerShell script.

PowerShell Script

The PowerShell script adds a Microsoft .NET class to a PowerShell session using Add-Type -typedefinition. It then reads the shellcode from the registry entry created by the VBScript and loads the shellcode into the virtually allocated memory space using the API ZwAllocateVirtualMemory and RtlMoveMemory. After that, the shellcode is launched inside powershell.exe using the API Callwindowprocw function.

The first 40 bytes of the shellcode are executable assembly instructions and the remaining bytes are encrypted. The first 40 bytes are responsible for decrypting the remaining part of the code, then the execution flow jumps into the decrypted part. During the next stage, the shellcode replaces the first 40 hex bytes with a no operation (NOP) instruction. This is done to avoid re-debugging of the code.

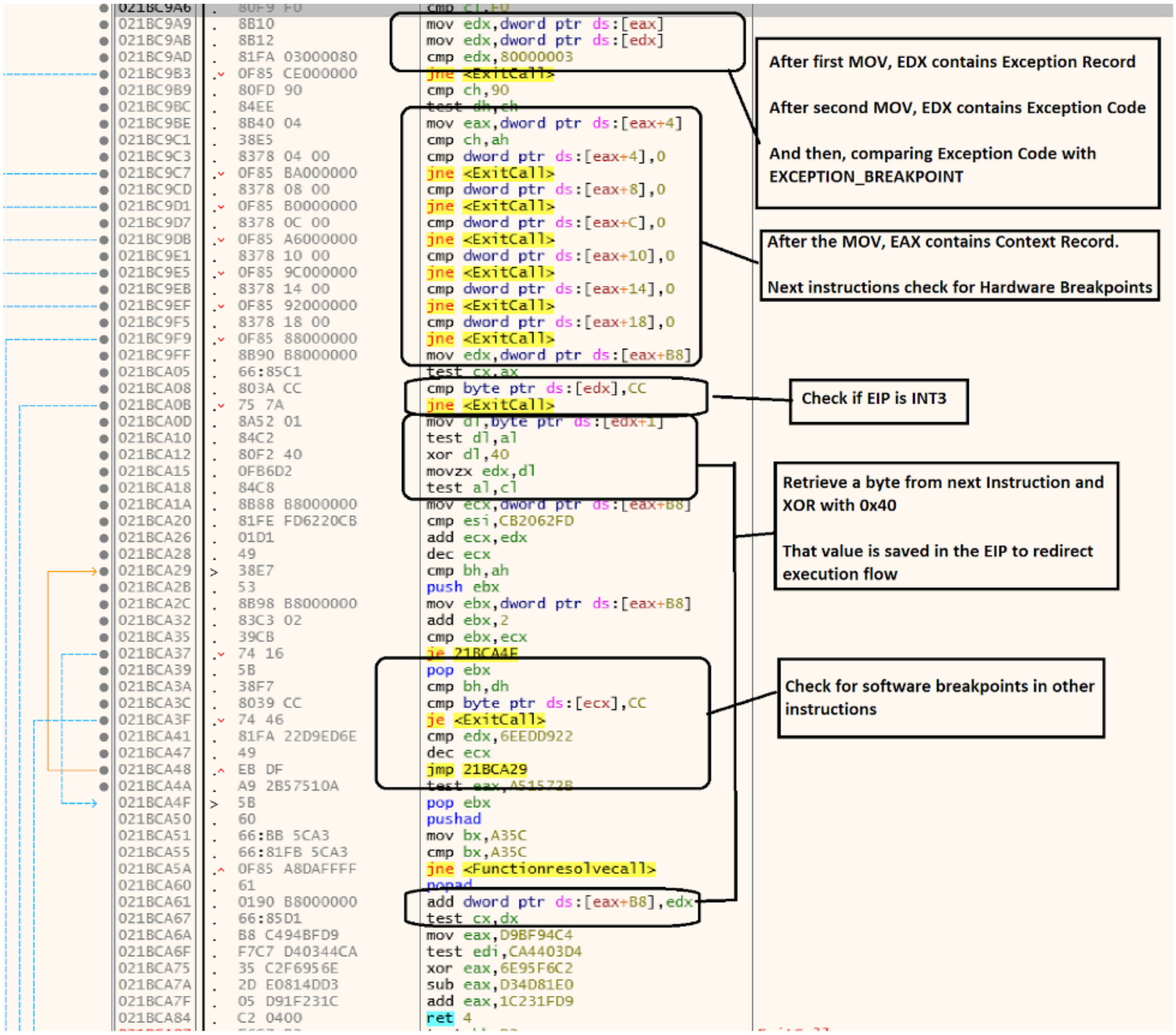
Initially, the shellcode traverses the process environment block (PEB) structure and fetches Ntdll to tap into exported NTAPI functions. The rest of the DLLs are loaded by using LoadLibrary. As previously covered by CrowdStrike, GuLoader uses the DJB2 algorithm to load APIs. The assembly code for DJB2 traverses through export functions of the required DLLs one by one, calculates the DJB2 hashes for each export API and then compares those with the hardcoded hash value. We covered the DJB2 algorithm in a previous blog post.

Anti-Analysis Techniques

The shellcode employs several anti-analysis and anti-debugging tricks at every step of execution, throwing an error message if the shellcode detects any known analysis of debugging mechanisms.

Anti-Debugging

GuLoader uses a vectored exception handler (VEH) to throw off researchers and make disassembly and debugging difficult by disrupting the normal flow of code execution to point the control flow to incorrect paths, raising exceptions that jump to other instructions. To add the exception, the shellcode uses the RtlAddVectoredExcepitionHandler API function.



```
021BC9A6 .  80F9 F0          cmp cl,F0
021BC9A9 .  8B10             mov edx,dword ptr ds:[eax]
021BC9AB .  8B12             mov edx,dword ptr ds:[edx]
021BC9AD .  81FA 03000080    cmp edx,80000003
021BC9B3 .v 0F85 CE000000    jne <ExitCall>
021BC9B9 .  80FD 90          cmp ch,90
021BC9BC .  84EE             test dh,ch
021BC9BE .  8B40 04          mov eax,dword ptr ds:[eax+4]
021BC9C1 .  38E5             cmp ch,ah
021BC9C3 .  8378 04 00       cmp dword ptr ds:[eax+4],0
021BC9C7 .v 0F85 BA000000    jne <ExitCall>
021BC9CD .  8378 08 00       cmp dword ptr ds:[eax+8],0
021BC9D1 .v 0F85 B0000000    jne <ExitCall>
021BC9D7 .  8378 0C 00       cmp dword ptr ds:[eax+C],0
021BC9DB .v 0F85 A6000000    jne <ExitCall>
021BC9E1 .  8378 10 00       cmp dword ptr ds:[eax+10],0
021BC9E5 .v 0F85 9C000000    jne <ExitCall>
021BC9EB .  8378 14 00       cmp dword ptr ds:[eax+14],0
021BC9EF .v 0F85 92000000    jne <ExitCall>
021BC9F5 .  8378 18 00       cmp dword ptr ds:[eax+18],0
021BC9F9 .v 0F85 88000000    jne <ExitCall>
021BC9FF .  8B90 B8000000    mov edx,dword ptr ds:[eax+B8]
021BCA05 .  66:85C1          test cx,ax
021BCA08 .  803A CC          cmp byte ptr ds:[edx],CC
021BCA0B .v 75 7A            jne <ExitCall>
021BCA0D .  8A52 01          mov dl,byte ptr ds:[edx+1]
021BCA10 .  84C2             test dl,al
021BCA12 .  80F2 40          xor dl,40
021BCA15 .  0FB6D2           movzx edx,dl
021BCA18 .  84C8             test al,cl
021BCA1A .  8B88 B8000000    mov ecx,dword ptr ds:[eax+B8]
021BCA20 .  81FE FD6220CB    cmp esi,CB2062FD
021BCA26 .  01D1             add ecx,edx
021BCA28 .  49               dec ecx
021BCA29 >  38E7             cmp bh,ah
021BCA2B .  53               push ebx
021BCA2C .  8B98 B8000000    mov ebx,dword ptr ds:[eax+B8]
021BCA32 .  83C3 02          add ebx,2
021BCA35 .  39CB             cmp ebx,ecx
021BCA37 .v 74 16            je 21BCA4F
021BCA39 .  5B               pop ebx
021BCA3A .  38F7             cmp bh,dh
021BCA3C .  8039 CC          cmp byte ptr ds:[ecx],CC
021BCA3F .v 74 46            je <ExitCall>
021BCA41 .  81FA 22D9ED6E    cmp edx,6EEDD922
021BCA47 .  49               dec ecx
021BCA48 .^ EB DF            jmp 21BCA29
021BCA4A .  A9 2B57510A      test eax,A515728
021BCA4F >  5B               pop ebx
021BCA50 .  60               pushad
021BCA51 .  66:BB 5CA3       mov bx,A35C
021BCA55 .  66:81FB 5CA3     cmp bx,A35C
021BCA5A .^ 0F85 A8DAFFFF    jne <Functionresolvecall>
021BCA60 .  61               popad
021BCA61 .  0190 B8000000    add dword ptr ds:[eax+B8],edx
021BCA67 .  66:85D1          test cx,dx
021BCA6A .  B8 C494BFD9      mov eax,D9BF94C4
021BCA6F .  F7C7 D40344CA    test edi,CA4403D4
021BCA75 .  35 C2F6956E      xor eax,6E95F6C2
021BCA7A .  2D E0814DD3      sub eax,D34D81E0
021BCA7F .  05 D91F231C      add eax,1C231FD9
021BCA84 .  C2 0400          ret 4
```

After first MOV, EDX contains Exception Record

After second MOV, EDX contains Exception Code

And then, comparing Exception Code with EXCEPTION_BREAKPOINT

After the MOV, EAX contains Context Record.

Next instructions check for Hardware Breakpoints

Check if EIP is INT3

Retrieve a byte from next Instruction and XOR with 0x40

That value is saved in the EIP to redirect execution flow

Check for software breakpoints in other instructions

Figure 1. Vector Exception Handler function (click to enlarge)

GuLoader performs a series of anti-debugging and anti-disassembling checks to detect the presence of breakpoints, usually associated with researchers analyzing its code execution flow.

For example, it extracts information from EXCEPTION_RECORD when it hits INT3 (0xCC) instruction and then it checks ExceptionCode from it. To determine if the VEH routine has been triggered by an INT3 instruction, it will check if the value matches 0x80000003 (e.g., EXCEPTION_BREAKPOINT). It then retrieves the DR registers from CONTEXT_RECORD to check if there are any HARDWARE breakpoints and it also checks the EIP (Extended Instruction Pointer) to see if it is equal to 0xCC. Looking for software breakpoints, GuLoader also checks for the presence of other 0xCC instructions in code and terminates execution if found (shown in Figure 1). If everything is as expected, the malware performs an XOR operation on the next byte after EIP and then replaces the EIP on CONTEXT with the new value, ensuring the execution flow will reach the correct address.

To bypass this check, to automatically jump to the next real address and avoid the VEH routine, we can use the following statement, inside the “command window” present at the bottom of x32 debugger, when the debugger reaches INT3 instruction (here the XOR value inside VEH was 0x40; it may be different in other samples):

```
eip=((ReadByte(eip+1)^0x40)+eip)
```

To avoid step-by-step replacement in the debugger, the following script can also be used to patch all of the INT3 instructions by replacing them with a JUMP to the real execution flow (copy the script below in the x32dbg “Script” tab):

```
call loop
loop:
mov $a, 0
findasm “int3”
cmp $result, $a
je exit
mov $temp, ref.addr($a)
mov $i, 0x40
xor [$temp+1], $i
sub [$temp+1], 2
1:[$temp] = 0xEB
jmp loop

exit:
ret
```

Breakpoint checks on APIs are performed before calling every API and if found, the shellcode terminates.

Using the NtsetInformationThread API is also an anti-debugging technique. The DJB2 algorithm loads the NtsetInformationThread API and passes the second parameter as 11 (corresponding to ThreadHideFromDebugger), which will crash the process when it runs from inside a debugger.

Anti-debugging via NtQueryInformationProcess enables GuLoader to check the presence of a remote debugger in its process. Our sample leverages the NtQueryInformationProcess API, by specifying ProcessDebugPort (0x7) as the second parameter. The loader checks for non-zero return values, which means the process is being debugged.

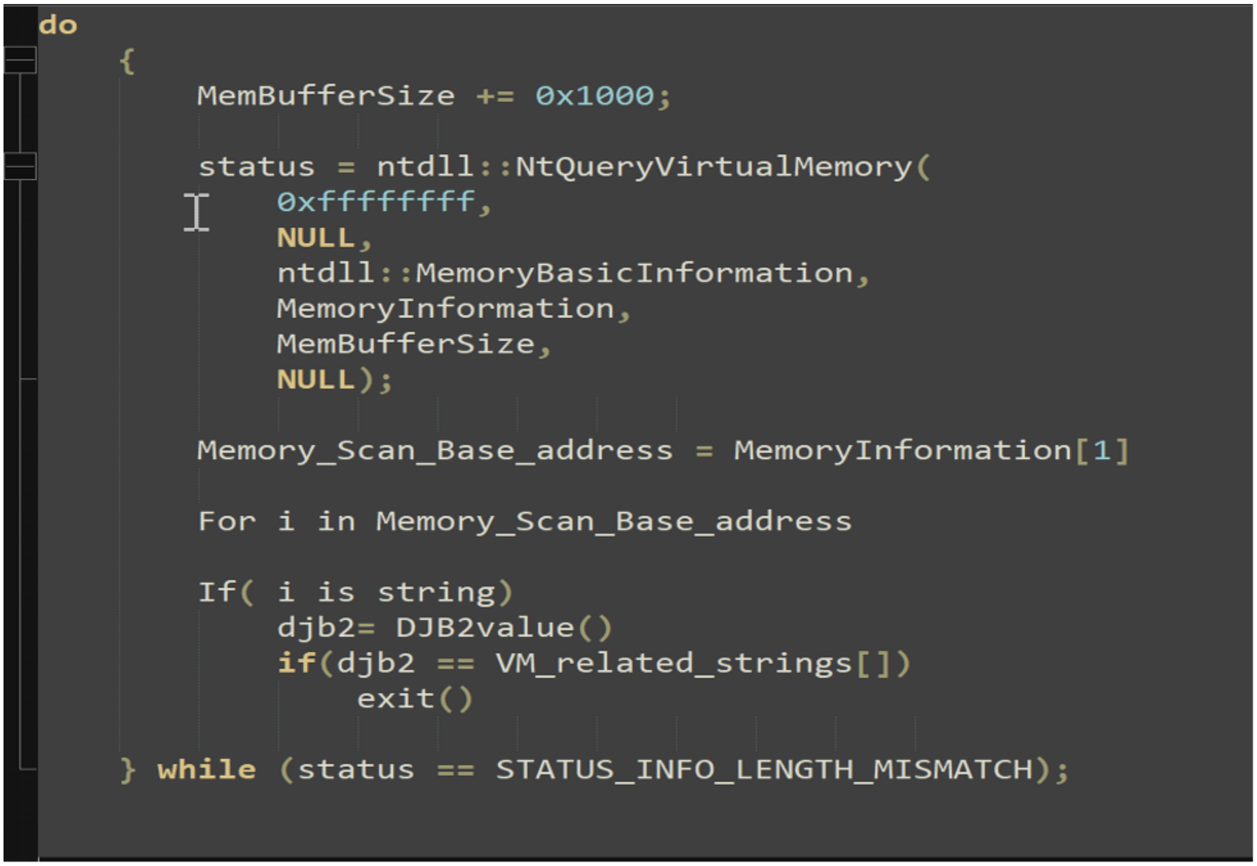
Anti-debugging via DbgBreakPoint and DbgUiRemoteBreakin allows GuLoader to patch two APIs in memory by leveraging the NtProtectVirtualMemory API to mark it writable for their addresses:

- DbgBreakPoint — by replacing with a NOP instruction
- DbgUiRemoteBreakin — by replacing with a random instruction

DJB2	API Name
4a082415	DbgBreakPoint
880bb688	DbgUiRemoteBreakin

Anti-Virtual Machine

What's different from previously analyzed GuLoader variants is that this shellcode performs **memory scanning for VMware-related string checks on every memory page** from the entire process memory. GuLoader uses NtQueryVirtualMemory API to scan the entire memory of the process to check if there are any Virtual Machine (VM)-related strings present.



```
do
    {
        MemBufferSize += 0x1000;

        status = ntdll::NtQueryVirtualMemory(
            0xffffffff,
            NULL,
            ntdll::MemoryBasicInformation,
            MemoryInformation,
            MemBufferSize,
            NULL);

        Memory_Scan_Base_address = MemoryInformation[1]

        For i in Memory_Scan_Base_address

        If( i is string)
            djb2= DJB2value()
            if(djb2 == VM_related_strings[])
                exit()

    } while (status == STATUS_INFO_LENGTH_MISMATCH);
```

Figure 2. Pseudocode showing how the Virtual Memory is scanned and how the Djb2 algorithm is used (click to enlarge)

This is implemented by calling API NtQueryVirtualMemory with the handle 0xffffffff (current process) to iteratively retrieve the base address of every page. The fourth parameter of this API is the MemoryInformation structure which contains information about a range of pages in the virtual address space of a process. A similar technique has been implemented in the past on VMDE project by hfiref0x, in which the author searches in memory for “Sandboxie” artifacts strings; in this case GuLoader searches for virtualization software traces.

If it finds any of the DJB2 values for a series of strings (i.e., VMSwitchUserControlClass, VM3DService Hidden Window, VMDisplayChangeControlClass, vmtoolsdControlWndClass, etc.), the shellcode throws an error message that it is running under a virtual environment and then terminates execution.

Using CPUID and rdtsc is a very common anti-debugging trick that involves using the read time-stamp counter (rdtsc) instruction to determine how many CPU ticks took place since the processor was reset. This is used as a timing check comparing the time required to execute two rdtsc instructions and then calling the CPUID instruction with EAX =1 to retrieve the process information, returning the output in the ECX registry. If the thirty-first bit of ECX is set, it is used as that the shellcode is running inside a potentially hostile environment (virtual machine).

The **use of the EnumWindows function** is also a popular anti-VM technique generally used to enumerate all top-level windows on the screen by passing the handle to each window. This API is used in the shellcode, counting the number of open windows inside the callback function. If the number is lower than 12, it will call the API TerminateProcess.

Enumerating device drivers also falls under the anti-VM category. GuLoader uses EnumDeviceDriver from (psapi.dll) and checks the presence of specific drivers and triggers an error if found. Shellcode fetches and calls two APIs from its DJB2 values — DADA7345 and CDAFD506, respectively EnumDeviceDriver and GetDeviceDriverBasename — to enumerate driver names. Every enumerated driver name’s DJB2 hash value is calculated. These hash values are then compared with hard-coded DJB2 hash values, which are actually VM-related device drivers.

DJB2 Value	Strings
9ba8433a	vmmouse.sys
d5360503	vm3dmp_loader.sys
D8FB0271	vm3dmp.sys
52eb67f8	vmusbmouse.sys

After bypassing all of the above tricks consecutively, it then loads addresses of several APIs as show in the table below:

DJB2 Value	API Name
c4835d68	NtsetContextthread
C45db42d	NtWriteVirtualMemory
D05D0AFC	ZwCreateSection
C101ddb2	NtMapViewOfSection
3b640034	NtsetInformationProcess
8ad0acb1	NtOpenFile
De797b11	NtClose
2334ac18	NtResumeThread
1a45d798	NtCreateThreadEx
9688DA44	CreateProcessInternalW

Scanning and enumerating installed software is a technique GuLoader uses to check for virtualization software installed as part of its anti-sandbox/anti-VM mechanism by loading APIs that match the DJB2 hash values 55fbd1cd (MsiGetProductInfoA) and AD5448 (MsiEnumProductsA). The shellcode enumerates the products using API MsiGetProductInfoA and checks if they match with a list of known software.

Service enumeration using the OpenScManager API establishes a connection to the service control manager on the machine and opens the specified service control manager database. It then enumerates service control manager database services using EnumServicesStatusA**.**

Process Hollowing

Process hollowing is a technique of executing arbitrary code in the address space of a separate live process by creating a process in a suspended state then unmapping/hollowing its memory, which can then be replaced with malicious code. In this case, the malware does not unmap an already mapped section on the remote process, but tries to add a new section and write the injected shellcode into it.

The below steps are followed for injection:

1. The shellcode first creates a suspended process by calling CreateProcessInternal.

2. It then calls `NtOpenFile` on `\\??\\C:\\Windows\\syswow64\\iertutil.dll`.

3. It does `NtCreateSection` on that file, where it will inject its malicious shellcode.

4. It then maps that section via NtMapViewofSection on the suspended process. **If this injection technique fails, it uses the following redundancy method:**

a. `NtAllocateVirtualMemory` by invoking the inline assembly instructions (without calling ntdll.dll, to bypass AV/EDR User Mode hooks) of that function, using the following assembly stub:

```
mov eax,18
mov edx,ntdll.77178850
call edx
ret 18
```

It uses `NtWriteProcessMemory` to copy the same shellcode onto that virtually allocated address.

5. It then uses API `NtGetContextThread` on remote thread of suspended process, by specifying the following flags to retrieve the registry values of that thread:

i. CONTEXT_CONTROL to retrieve the registers ESP, EIP, FLAGS, BP

ii. CONTEXT_INTEGER to retrieve the registers AX, BX, CX, DX, SI, DI

iii. CONTEXT_SEGMENTS to retrieve the registers DS, ES, FS, GS

6. The retrieved `CONTEXT` is used to manipulate registers by calling the NTAPI `NtSetContextThread` to set the `EAX` register to the address of shellcode (EIP points to `RtlUserThreadStart`, which will jump to new `EAX`).

7. Finally, the malware calls the native API `NtResumeThread` to resume the process and execute the shellcode in the new process.

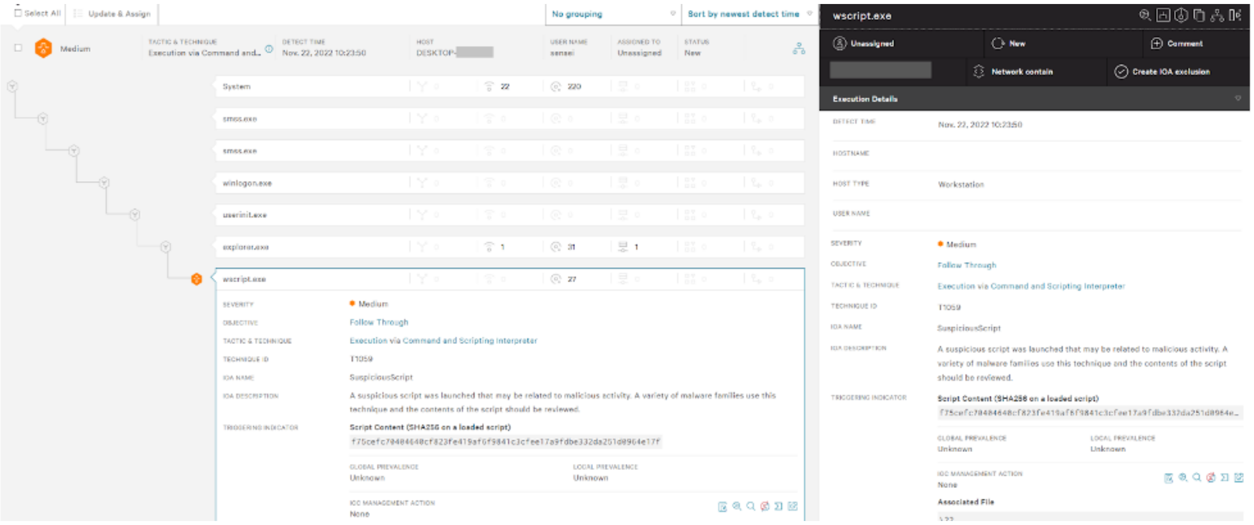
Final Shellcode

After injection, the shellcode re-executes all the anti-analysis steps mentioned above, and then decrypts the URL to retrieve the last payload from https[:]//biropem[.]papuabaratprov[.]go[.]id/bin_fXZOFMVq248[.]bin. It loads the API to manage the internet connection and connects to a URL to download the first level of payload, which then gets decrypted by the shellcode and results in dropping the Remcos malware.

DJB2	API Name
C6e89145	InternetOpenA
9f39811c	InternetSetOptionA

292652cc	InternetOpenUrlA
F978A052	InternetReadFile
71167D2C	InternetCloseHandle

How the CrowdStrike Falcon Platform Protects Against GuLoader



Select All | Update & Assign | No grouping | Sort by newest detect time

Medium | TACTIC & TECHNIQUE: Execution via Command and... | DETECT TIME: Nov. 22, 2022 10:23:50 | HOST: DESKTOP-[redacted] | USER NAME: sensei | ASSIGNED TO: Unassigned | STATUS: New

System	0	22	220	0	0	0
smss.exe	0	0	0	0	0	0
smss.exe	0	0	0	0	0	0
winlogon.exe	0	0	0	0	0	0
userinit.exe	0	0	0	0	0	0
explorer.exe	0	1	31	1	0	0
wscript.exe	0	0	27	0	0	0

SEVERITY: Medium
OBJECTIVE: Follow Through
TACTIC & TECHNIQUE: Execution via Command and Scripting Interpreter
TECHNIQUE ID: T1059
IOA NAME: SuspiciousScript
IOA DESCRIPTION: A suspicious script was launched that may be related to malicious activity. A variety of malware families use this technique and the contents of the script should be reviewed.
TRIGGERING INDICATOR: Script Content (SHA256 on a loaded script)
f75cefc70404640cf823fe419af6f9841c3cfee17a9fdbe332da251d0964e17f
GLOBAL PREVALENCE: Unknown
LOCAL PREVALENCE: Unknown
IOC MANAGEMENT ACTION: None

wscript.exe

[redacted]

Execution Details

DETECT TIME: Nov. 22, 2022 10:23:50
HOSTNAME:
HOST TYPE: Workstation
USER NAME:
SEVERITY: Medium
OBJECTIVE: Follow Through
TACTIC & TECHNIQUE: Execution via Command and Scripting Interpreter
TECHNIQUE ID: T1059
IOA NAME: SuspiciousScript
IOA DESCRIPTION: A suspicious script was launched that may be related to malicious activity. A variety of malware families use this technique and the contents of the script should be reviewed.
TRIGGERING INDICATOR: Script Content (SHA256 on a loaded script)
f75cefc70404640cf823fe419af6f9841c3cfee17a9fdbe332da251d0964e…
GLOBAL PREVALENCE: Unknown
LOCAL PREVALENCE: Unknown
IOC MANAGEMENT ACTION: None
Associated File
\??

Click to enlarge

GuLoader remains a dangerous threat that’s been constantly evolving with new methods to evade detection. The Falcon platform uses behavior-based detection capabilities to automatically detect and prevent GuLoader early in the attack chain by identifying the initial VBScript loader, preventing its execution.

This recent analysis performed by CrowdStrike on GuLoader now offers a complete picture of all the DJB2 hash values used for APIs. This type of threat research enables CrowdStrike to leverage expert human intelligence and augment its machine learning and behavior-based detection capabilities to stop breaches.

Indicators of Compromise (IOCs)

File	SHA256
GuLoader	f75cefc70404640cf823fe419af6f9841c3cfee17a9fdbe332da251d0964e17f

Appendix

The following table contains the complete list of additional DJB2 hash values for APIs as used by GuLoader.

DJB2 Value	API Name
DADA7345	EnumDeviceDriver
CDAFD506	GetDeviceDriverBasename
c4835d68	NtsetContextthread
C45db42d	NtWriteVirtualMemory
D05D0AFC	ZwCreateSection
C101ddb2	NtMapViewOfSection
3b640034	NtsetInformationProcess
8ad0acb1	NtOpenFile
De797b11	NtClose
2334ac18	NtResumeThread
1a45d798	NtCreateThreadEx
9688DA44	CreateProcessInternalW
55fbd1cd	MsiGetProductInfoA
AD5448	MsiEnumProductsA
4a082415	DbgBreakPoint
880bb688	DbgUiRemoteBreakin
C6e89145	InternetOpenA
9f39811c	InternetSetOptionA
292652cc	InternetOpenUrlA
F978A052	InternetReadFile
71167D2C	InternetCloseHandle

Additional Resources

- *Learn how the powerful CrowdStrike Falcon® platform provides comprehensive protection across your organization, workers and data, wherever they are located.*
- *Get a full-featured free trial of CrowdStrike Falcon Prevent™ and see for yourself how true next-gen AV performs against today's most sophisticated threats.*