

GitHub - knight0x07/NailaoLoader-Hiding-Execution-Flow: NailaoLoader: Hiding Execution Flow via Patching

By knight0x07

Archived: 2026-04-05 17:04:45 UTC

NailaoLoader: Hiding Execution Flow via Patching

Background

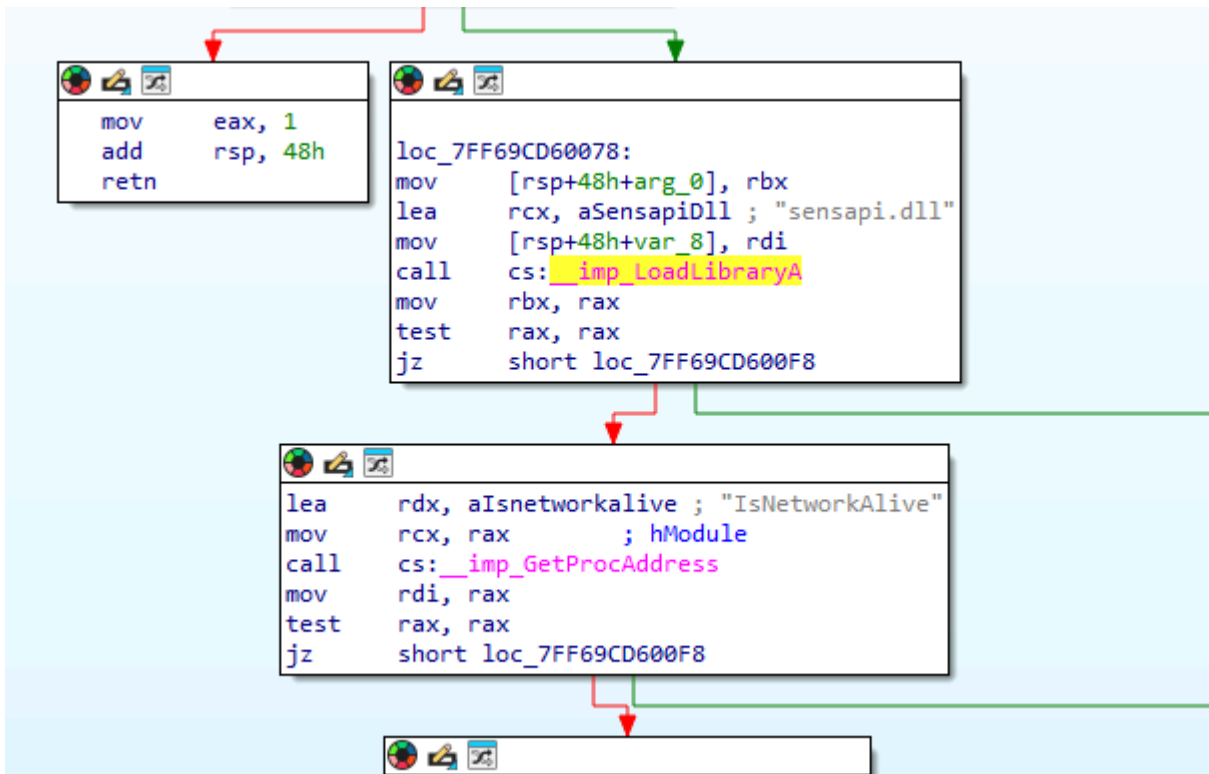
The threat actors were seen using Windows Management Instrumentation (WMI) to transfer the following three files (and execute usysdiag.exe) to each machine by executing a script that targeted a list of local IP addresses:

- usysdiag.exe
- sensapi.dll
- usysdiag.exe.dat

Analysis

The usysdiag.exe (**Huorong Sysdiag Helper - Huorong Internet Security**) which is a valid signed executable by "**Beijing Huorong Network Technology Co. Ltd.**" - Chinese endpoint security solutions provider is initially executed.

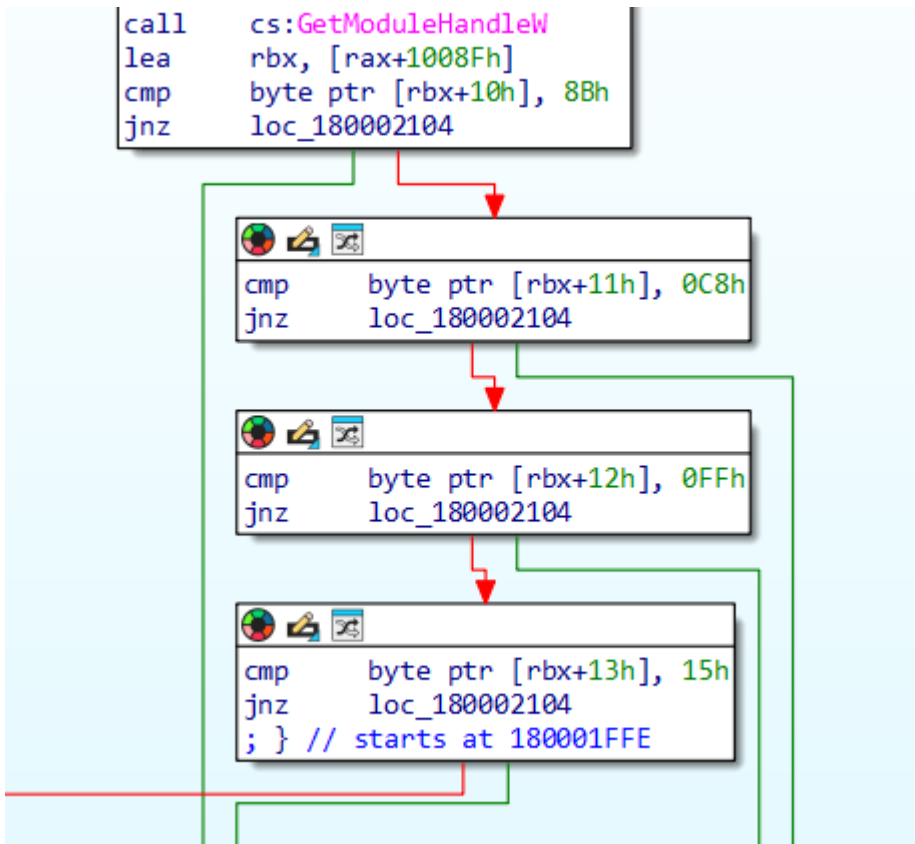
The usysdiag.exe calls a function which calls **LoadLibraryA()** to load "**sensapi.dll**" in its virtual address space and then calls **GetProcAddress()** function to fetch address to the **IsNetworkAlive()** function exported by "**sensapi.dll**".



But in our case, the **NailaoLoader** "sensapi.dll" is been **side-loaded** instead of the legitimate DLL as it is present alongside the "usysdiag.exe" in the same directory and then once the NailaoLoader DLL is loaded the **DllMain()** function of the malicious DLL is called.

The NailaoLoader's **DllMain()** function initially calls **GetModuleHandleW()** with **lpModuleName = NULL** which retrieves the handle (**image base address - eg. 0x00007FF69CD50000**) of the calling process "usysdiag.exe" and then it verifies a bytes sequence at a specific offset in the ****.text section**** of usysdiag.exe. Following is the byte sequence which is verified:

- At <usysdiag_image_base_address> + 0x1008F + 0x10 compares byte: 0x8B
- At <usysdiag_image_base_address> + 0x1008F + 0x11 compares byte: 0xC8
- At <usysdiag_image_base_address> + 0x1008F + 0x12 compares byte: 0xFF
- At <usysdiag_image_base_address> + 0x1008F + 0x13 compares byte: 0x15



The **NailaoLoader** in this case checks for the following instructions `mov ecx, eax ; call qword ptr (8B C8 FF 15)` at the given offset and interestingly the offset into the `usysdiag`'s `.text` section is basically in the same initial function itself which called the `LoadLibraryA()` to load `"sensapi.dll"`.

FF15 71F20300	<code>call qword ptr ds:[<LoadLibraryA>]</code>
48:8BD8	<code>mov rbx,rax</code>
48:85C0	<code>test rax,rax</code>
74 61	<code>je usysdiag.7FF69CD600F8</code>
48:8D15 0ADF0400	<code>lea rdx,qword ptr ds:[7FF69CDADFA8]</code>
48:8BC8	<code>mov rcx,rax</code>
FF15 79F20300	<code>call qword ptr ds:[<GetProcAddress>]</code>

If the byte sequence does not match the `DllMain()` returns and the `NailaoLoader` does not execute the malicious code. Therefore the following executable `"usysdiag.exe"` is required in order to execute the malicious code of the `NailaoLoader`.

Further if the bytes match then it performs following actions:

- calls the `ret_virtual_protect_addr()` function which returns the address of `VirtualProtect()` function by firstly allocating a 100 byte buffer and then performing `cus_memcpy()` to copy the string `"VirtualProtect"` byte by byte into the allocated buffer.

```
call    alloc_mem
lea     r8, aV          ; "V"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aI          ; "i"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aR          ; "r"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aT          ; "t"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aU          ; "u"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aA          ; "a"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, asc_180017894 ; "l"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aP          ; "P"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aR          ; "r"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aO          ; "o"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aT          ; "t"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aE          ; "e"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aC          ; "c"
mov     edx, 100h
lea     rcx, [rsp+138h+ProcName]
call    mem_cpy
lea     r8, aT          ; "t"
```

Then it calls **GetModuleHandleA()** where **lpModuleName = kernel32** to get the handle to **kernel32.dll** and then calls **GetProcAddress()** with the handle to kernel32.dll and lpProcName as **VirtualProtect** to get the address of **VirtualProtect()** and returns the address to it.

- Then it calls the **VirtualProtect()** function to change the page protection to **PAGE_READWRITE** of the memory region in usysdiag.exe's .text section which consists of the **initial function which called LoadLibraryA() to load "sensapi.dll"**

0x7ff69cd50000	468 kB	Image	WCX	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cd50000	4 kB	Image: Commit	R	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cd51000	60 kB	Image: Commit	RX	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cd60000	4 kB	Image: Commit	RW	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cd61000	248 kB	Image: Commit	RX	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cd9f000	96 kB	Image: Commit	R	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cdb7000	16 kB	Image: Commit	RW	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cddb000	20 kB	Image: Commit	R	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cdc0000	4 kB	Image: Commit	WC	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe
0x7ff69cdc1000	16 kB	Image: Commit	R	C:\Users\knigh\Desktop\241108-m6lpeatmdk_pw_infected\usysdiag.exe.dat\Comms\usysdiag.exe

NOW in order to execute the **load_decrypt_exec_locker_func()** function which is the main function which reads the encrypted usysdiag.exe.dat file from the disk, decrypts it using a XOR key and then maps the decrypted NailaoLocker binary in memory and transfers the control flow to the binary's entrypoint.

The **NailaoLoader** patches the following instructions which are just after the call to LoadLibraryA("sensapi.dll") in the initial function that we say was called by usysdiag.exe when executed.

```
mov rbx, rax
test rax, rax
```

Patched to:

```
mov rax, sensapi.7FFA8D1E1DF0 [address of load_decrypt_exec_locker_func() function]
call rax
```

If we compare both the unpatched and patched versions of the initial function called by usysdiag.exe calling the LoadLibraryA() to load Sensapi.dll we can clearly see the difference:

Function Before Patching:				
00007FF69CD60078	48:895C24 50	mov qword ptr ss:[rsp+50],rbx	00007FF69CDADF98:"sensapi.dll"	
00007FF69CD6007D	48:8D0D 14DF0400	lea rcx,qword ptr ds:[7FF69CDADF98]		
00007FF69CD60084	48:897C24 40	mov qword ptr ss:[rsp+40],rdi		
00007FF69CD60089	FF15 71F20300	call qword ptr ds:[<LoadLibraryA>]		
00007FF69CD6008F	48:8BD8	mov rbx, rax	un-patched instructions	
00007FF69CD60092	48:85C0	test rax, rax		
00007FF69CD60095	74 61	je usysdiag.7FF69CD600F8	00007FF69CDADF98:"IsNetworkAlive"	
00007FF69CD60097	48:8D15 0ADF0400	lea rdx,qword ptr ds:[7FF69CDADFA8]		
00007FF69CD6009E	48:8BC8	mov rcx, rax	patched instructions	
00007FF69CD600A1	FF15 79F20300	call qword ptr ds:[<GetProcAddress>]		
00007FF69CD600A7	48:8BF8	mov rdi, rax		
00007FF69CD600AA	48:85C0	test rax, rax		
00007FF69CD600AD	74 49	je usysdiag.7FF69CD600F8		
00007FF69CD60078	48:895C24 50	mov qword ptr ss:[rsp+50],rbx		00007FF69CDADF98:"sensapi.dll"
00007FF69CD6007D	48:8D0D 14DF0400	lea rcx,qword ptr ds:[7FF69CDADF98]		
00007FF69CD60084	48:897C24 40	mov qword ptr ss:[rsp+40],rdi		
00007FF69CD60089	FF15 71F20300	call qword ptr ds:[<LoadLibraryA>]		
00007FF69CD6008F	48:B8 F01D1E8DFA7F000	mov rax, sensapi.7FFA8D1E1DF0	Addr of load_decrypt_exec_locker_func()	
00007FF69CD60099	FFD0	call rax		
00007FF69CD6009B	DF0400	rld word ptr ds:[rax+rax]	patched instructions	
00007FF69CD6009E	48:8BC8	mov rcx, rax		
00007FF69CD600A1	FF15 79F20300	call qword ptr ds:[<GetProcAddress>]		
00007FF69CD600A7	48:8BF8	mov rdi, rax		
00007FF69CD600AA	48:85C0	test rax, rax		
00007FF69CD600AD	74 49	je usysdiag.7FF69CD600F8		

Disassembled code of NailaoLoader patching the instructions:

```
movzx  eax, byte ptr [rsp+48h+var_20+1]
movzx  edx, byte ptr [rsp+48h+var_20+5]
mov    word ptr [rbx], 0B848h
mov    [rbx+3], al
movzx  eax, byte ptr [rsp+48h+var_20+2]
mov    [rbx+4], al
movzx  eax, byte ptr [rsp+48h+var_20+3]
mov    [rbx+7], dl
movzx  edx, byte ptr [rsp+48h+var_20+6]
mov    [rbx+5], al
movzx  eax, byte ptr [rsp+48h+var_20+4]
mov    [rbx+8], dl
movzx  edx, byte ptr [rsp+48h+var_20+7]
mov    [rbx+6], al
mov    [rbx+2], sil
mov    [rbx+9], dl
mov    word ptr [rbx+0Ah], 0D0FFh
```

So now whenever the **LoadLibraryA()** function trying to load the "sensapi.dll" ; returns the next instruction called would be the patched instructions which will move the address of **load_decrypt_exec_locker_func()** function into **rax** and then call **rax** i.e call the **load_decrypt_exec_locker_func()** function

This technique helps in hiding the execution flow of the Loader as the load-decrypt-execute function is not called from the malicious NailaoLoader DLL itself.

Then it again calls **VirtualProtect()** and sets the page protection of the same memory region of the .text section to **PAGE_EXECUTE_READ** (back to the old page protection) and then returns from the DllMain() of the sensapi.dll.

Further when the LoadLibraryA("sensapi.dll") function called by usysdiag.exe returns back it then executes the patched instructions and calls the **load_decrypt_exec_locker_func()** which then further executes the **NailaoLocker!**

```
mov rax,sensapi.7FFA8D1E1DF0 [addr of load_decrypt_exec_locker_func() function]
call rax
```

This is how the NailaoLoader hides the Execution Flow via Patching Instructions in order to run the load-decrypt-execute function which further executes the NailaoLocker.

Campaign Reference: <https://www.orange cyberdefense.com/global/blog/cert-news/meet-nailaolocker-a-ransomware-distributed-in-europe-by-shadowpad-and-plugx-backdoors>