Stealing the LIGHTSHOW (Part Two) — LIGHTSHIFT and LIGHTSHOW

mandiant.com/resources/blog/lightshift-and-lightshow

In part one on <u>North Korea's UNC2970</u>, we covered <u>UNC2970</u>'s tactics, techniques and procedures (TTPs) and tooling that they used over the course of multiple intrusions. In this installment, we will focus on how UNC2970 utilized Bring Your Own Vulnerable Device (BYOVD) to further enable their operations.

During our investigation, Mandiant consultants identified most of the original compromised hosts, targeted by UNC2970, contained the files %temp%\<random>_SB_SMBUS_SDK.dll and suspicious drivers, created around the same time on disk.

At the time Mandiant initially identified these files, we were unable to determine how they were dropped or the exact use for these files. It wasn't until later in the investigation, during analysis of a forensic image, where the pieces started falling into place. A consultant noticed multiple keyword references to the file C:\ProgramData\USOShared\Share.DAT (MD5: def6f91614cb47888f03658b28a1bda6). Upon initial glance at the Forensic Image, this file was no longer on disk. However, Mandiant was able to recover the original file, and the initial analysis of the sample found that Share.DAT was a XORed data blob, which was encoded with the XOR key 0x59.

The decoded payload (MD5: 9176f177bd88686c6beb29d8bb05f20c), referred to by Mandiant as <u>LIGHTSHIFT</u>, is an in-memory only dropper. The LIGHTSHIFT dropper distributes a payload (MD5: ad452d161782290ad5004b2c9497074f) that Mandiant refers to as <u>LIGHTSHOW</u>. Once loaded into memory, LIGHTSHIFT invokes the exports <u>Create</u> then <u>Close</u> in that order. The response from <u>Close</u> is written as a hex formatted address to the file <u>C:\Windows\windows.ini</u>.

	IDA View-A) P	seudocode-A	13	0	Hex View-1	
•	.text:0000000180001090	8	test	rax, rax				
	.text:0000000180001093		jnz	short loc_1800	010A8			
1.0	.text:0000000180001095		mov	rcx, rbp	; hMem			
	.text:0000000180001098		call	cs:LocalFree				
- C 🐮	.text:000000018000109E		mov	eax, 0FFFF0000	h			
	.text:00000001800010A3		jmp	loc_18000118C				
1	.text:00000001800010A8	;						
- 6	.text:00000001800010A8							
1	.text:00000001800010A8	loc_18000	10A8:		; CODE	XREF: Star	tAddress+931j	
40	.text:00000001800010A8		lea	rdx, aCreate	; "Crea	ate"		
•	.text:00000001800010AF		mov	rcx, rax				
	.text:00000001800010AF	; } // st	arts at 18000	1000				
	.text:00000001800010B2							
	.text:0000000180001082	loc_18000	10B2:		; DATA	XREF: .rda	ta:000000018000C62840	
	.text:00000001800010B2				; .rdat	ta:00000001	8000C63C4o	
	.text:00000001800010B2	; _unwir	id { //GSHa	IndlerCheck				
•	.text:00000001800010B2	e de la co ncense e G	mov	[rsp+78h+arg_0], rsi			
	.text:00000001800010BA		call	sub_180001AB0	0.00			
	.text:00000001800010BF		lea	rdx, aClose	; "Clos	se"		
•	.text:00000001800010C6		mov	rcx, rbx				
	.text:00000001800010C9		mov	rdi, rax				
•	.text:00000001800010CC		call	sub_180001AB0				
	.text:00000001800010D1		mov	rcx, cs:qword_	18008F2F6	3		
	.text:00000001800010D8		mov	rsi, rax				
•	.text:00000001800010DB		call	rdi				
	.text:00000001800010DD		test	eax, eax				
· · · ·	.text:00000001800010DF		jz	short loc_1800	010E8			
	.text:00000001800010E1		mov	esi, 0F0000004	h			
10 •	.text:00000001800010E6		jmp	short loc_1800	010EC			
0.000	.text:00000001800010E8	;						
20	.text:00000001800010E8							
L. and	.text:00000001800010E8	loc_18000	10E8:		; CODE	XREF: Star	tAddress+DF1j	
	.text:00000001800010E8		call	rsi				
	.text:00000001800010EA		mov	esi, eax				
	.text:00000001800010EC							
	.text:00000001800010EC	loc_18000	10EC:		; CODE	XREF: Star	tAddress+E61j	
	.text:00000001800010EC		mov	rcx, rbx	; lpMer	n		
	.text:00000001800010EF		call	sub_1800018D0				
	.text:00000001800010F4		xor	eax, eax				
	.text:00000001800010F6		mov	rdi, rbp				
	.text:00000001800010F9		mov	ecx, 7FF00h				
	.text:00000001800010FE		xor	ebp, ebp				
	.text:0000000180001100		lea	r8d, [rax+2]	; dwSha	areMode		
	.text:0000000180001104		rep st	osb				
	.text:0000000180001106		mov	[rsp+78h+hTemp	lateFile], rbp ; h1	TemplateFile	
	.text:000000018000110B		lea	rcx, FileName	; "C:\\	Windows \\w	vindows.ini"	
	.text:0000000180001112		xor	r9d, r9d	; lpSee	curityAttri	butes	
	.text:0000000180001115		mov	edx, 40000000	; dwDe:	siredAccess	Constraint Int Charge and	
	.text:000000018000111A		mov	[rsp+78h+dwFla	gsAndAtte	ributes], 8	Wh ; dwFlagsAndAttrib	outes
	.text:0000000180001122		mov	[rsp+78h+dwCre	ationDisp	position],	2 ; dwCreationDisposi	tion
	.text:000000018000112A		call	cs:CreateFileW	1			
	.text:0000000180001130		mov	rbx, rax				
	.text:0000000180001133		cmp	rax, OFFFFFFFF	FFFFFFFF	r		
	.text:0000000180001137		jz	short loc_1800	01182			
	.text:0000000180001139		lea	rdx, Format	; "%08)	<"		
	.text:0000000180001140		lea	rcx, [rsp+78h+	Buffer]	; Buffer		
	.text:0000000180001145		mov	r8d, esi				
	.text:0000000180001148		call	sprintf				
1	++ · 00000001000011400			0	recenter			

Figure 1: LIGHTSHIFT preparing to load LIGHTSHOW

LIGHTSHOW is a utility that makes use of two primary anti-analysis techniques used to hinder both dynamic and static analysis. To deter static analysis, LIGHTSHOW was observed being packed by VM-Protect. In an effort to thwart dynamic analysis, LIGHTSHOW is targeted to a specific host and requires a specific SHA256 hash corresponding to a specific computer name or the sample will not fully execute. Once FLARE completed the analysis of LIGHTSHOW, we were able to understand how the files %temp%\ <random>_SB_SMBUS_SDK.dll and drivers were created on disk. LIGHTSHOW is a utility that was used by UNC2970 to manipulate kernel data-structures and represents an advancement in DPRK's capabilities to evade detection. To accomplish this, LIGHTSHOW drops a legitimate version of a driver with <u>known vulnerabilities</u>, with a SHA256 hash of

175eed7a4c6de9c3156c7ae16ae85c554959ec350f1c8aaa6dfe8c7e99de3347 to C:\Windows\System32\Drivers with one of the following names chosen at random and appended with mgr :

- circlass
- dmvsc
- hidir
- isapnp
- umpass

LIGHTSHOW then creates the registry key

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<service name> where <service name> is the same as the chosen filename without appended mgr . It then creates a registry key with the value name ImagePath , which points to the path of the driver. The sample then loads the driver using NtLoadDriver . LIGHTSHOW drops and loads a dummy DLL %temp%\<random>_SB_SMBUS_SDK.dll to register itself to the driver as a legitimate caller.

Using the vulnerable driver, LIGHTSHOW can perform arbitrary read and write operations to kernel memory. LIGHTSHOW uses this read/write primitive to patch different kernel routines, which are related to the type of facilities an Endpoint Detection and Response (EDR) software may use, to enable evasion of said EDR software. After the read and write operations to kernel memory, the sample unloads and deletes <code>%temp%\<random\>_SB_SMBUS_SDK.dll</code>.

Examining the chain of execution, we see further obfuscation techniques being employed in LIGHTSHOW. UNC2970 has a concerted effort towards obfuscation and employs multiple methods to do this throughout the entire chain of delivery and execution.

	IDA View-A	×	Ō	Hex View-1	×	A	Structures	×
	.text:000007FEF1551010	var 7EC	= bv	te ptr -7ECh				
	.text:000007FEF1551010	var 7E0	= WO	rd ptr -7E0h				
	.text:000007FEF1551010	var 10	= aw	ord ptr -10h				
	.text:000007FEF1551010	var s0	= bv	tentr 0				
	.text:000007FEF1551010	arg 0	= aw	ord ptr 10h				
	.text:000007FEF1551010	arg 8	= QW	ord ptr 18h				
	.text:000007FEF1551010	arg 10	= aw	ord ptr 20h				
	.text:000007FEF1551010	arg 18	= QW	ord ptr 28h				
	.text:000007FEF1551010	0	4.					
	.text:000007FEF1551010	: unwind {	// GS	HandlerCheck				
2	.text:000007FEF1551010	·	mov	[rsp-8+arg 10], rbx				
	.text:000007FEF1551015			0_ 1/				
	.text:000007FEF1551015	loc 7FEF1551	015:	; DATA 3	KREF: sub 7FE	F157D	4F3:loc 7FEF1573383↓o	
1	.text:000007FEF1551015	-	mov	[rsp-8+arg 18], rdi	-		-	
1	.text:000007FEF155101A		push	rbp				
•	.text:000007FEF155101B		lea	rbp, [rsp-780h]				
1	.text:000007FEF1551023		sub	rsp, 880h				
•	.text:000007FEF155102A		mov	rax, cs:gword_7FEF15620	30			
1	.text:000007FEF1551031		xor	rax, rsp ; Non-z	ero XOR			
1	.text:000007FEF1551034		mov	[rbp+780h+var 10], rax				
1	.text:000007FEF155103B		mov	rbx, rcx				
1	.text:000007FEF155103E		lea	<pre>rcx, [rbp+780h+var_7E0+</pre>	1]			
1	.text:000007FEF1551042		xor	edx, edx				
1	.text:000007FEF1551044		mov	r8d, 7CFh				
1	.text:000007FEF155104A		mov	byte ptr [rbp+780h+var_	7E0], 0			
1	.text:000007FEF155104E		call	sub_7FEF155A3E0				
1	.text:000007FEF1551053		mov	r11, gs:30h				
1	.text:000007FEF155105C		mov	rdi, [r11+60h]				
1	.text:000007FEF1551060		mov	[rsp+880h+var_808], 6C6	4746Eh ; ntdl	1		
	.text:000007FEF1551068		mov	[rsp+880h+var_804], 6Ch	; '1'			
	.text:000007FEF155106F		mov	[rbp+780h+var_800], 755	1744Eh ; NtQu	eryVi	rtualMemory	
	.text:000007FEF1551076		mov	[rbp+780h+var_7FC], 567	97265h			
	.text:000007FEF155107D							
	.text:000007FEF155107D	10C_7FEF1551	07D:	; DATA .	KREF: sub_7FE	F157B	49C:1oc_7FEF15783C0↓o	
	.text:00000/FEF15510/D			; sub_/	-EF15//53A+3/	44↓0		
•	.text:000007FEF155107D		mov	[rbp+/60n+var_/r6], /5//	47269n			
•	.text:000007FEF1551064		mov	[rbp+760n+var_7r4], 654				
•	. LEXC:000007FEF155100D		mov	[rbp+780htvar_7F0], 797.	2010011			
•	text:000007EEE1551092		test	[Dp+/boiltval_/cc]; 0				
•	text:000007FEF1551090		inz	short loc 7EEE15510A2				
5	text:000007EE1551099		yor	83V 83V				
2	text:000007EEE155109D		imp	loc 7EEE1551183				
	.text:000007FEF15510A2	:						
	.text:000007FEF15510A2	·						
	.text:000007FEF15510A2	loc 7FEF1551	0A2:	; CODE	KREF: Create+	89 †i		
2	.text:000007FEF15510A2	-	lea	rcx, [rsp+880h+var 808]	; ntdll			
	.text:000007FEF15510A2	; } // start	s at 7FE	F1551010				
	.text:000007FEF15510A7							
	.text:000007FEF15510A7	loc_7FEF1551	0A7:	; DATA :	KREF: .rdata:	00000	7FEF15608A4↓o	
	.text:000007FEF15510A7			; .rdat	a:000007FEF15	6 0 8B8	ŧo	
, I	.text:000007FEF15510A7	;unwind {	//GSI	HandlerCheck				
	.text:000007FEF15510A7		mov	[rsp+880h+arg_8], rsi				
	.text:000007FEF15510AF		call	sub_7FEF1579D0E				
	.text:000007FEF15510B4		xchg	eax, esp				
	.text:000007FEF15510B5		lea	rdx, [rbp+780h+var_800]	; NtQueryVir	tualM	emory	
,	.text:000007FEF15510B9		mov	rcx, rax				
1	.text:00000/FEF15510BC		push	PCX				
	000004BC 000007FEF15510B0	Creste+AC	(Synchror	ized with Hey View-1)				

Figure 2: LIGHTSHOW Obfuscation

LIGHTSHOW is another example of tooling that looks to capitalize on the technique of BYOVD. BYOVD is a technique that utilizes the abuse of legitimate and trusted, but vulnerable drivers, to bypass kernel level protections. This technique has been utilized by adversaries ranging from financial actors, such as <u>UNC3944</u>, to espionage actors like UNC2970, which shows its usefulness during intrusion operations. AHNLab recently released a <u>report</u> on activity tracked as Lazarus Group that focused largely on the use of BYOVD. While Mandiant did not observe the hashes included in the AHNLab report, the use of <u>SB_SMBUS_SDK.dll</u> as well as other similarities, such as the exported functions <u>Create</u> and <u>Close</u>, indicate an overlap between the activity detailed in this blog post and those detailed by AHNLab. Throughout several incidents we responded to in 2022 that involved UNC2970, we observed them utilizing a small set of vulnerable drivers. This includes the <u>Dell DBUtil 2.3</u> and the ENE Technology device drivers. UNC2970 utilized both of these drivers in an attempt to evade detection. These two drivers, and many more, are found in the <u>Kernel Driver Utility</u> (KDU) toolkit. With this in mind, it is likely that we will continue to see UNC2970 abuse vulnerable drivers from other vendors.

Mandiant has worked to detect and mitigate BYOVD techniques for a <u>number of years</u> and has worked closely with industry allies to report vulnerabilities when discovered. During research being carried out on UNC2970 we discovered a <u>vulnerable driver</u> that the actor had access to, but did not know was vulnerable - essentially making it a Oday in the wild but not being actively exploited. This was verified through our Offensive Task Force who subsequently carried out a notification to the affected organization and reported the vulnerability to MITRE, which was assigned <u>CVE-2022-42455</u>.

Outlook and Implications

Mandiant continues to observe multiple threat actors utilizing BYOVD during intrusion operations. Because this TTP provides adversaries an effective means to bypass and mitigate EDR, we assess that it will continue to be utilized and adapted into actor tooling. The continued targeting of security researchers by UNC2970 also provides an interesting way that the group can potentially continue to expand their toolset to gain an upper hand with BYOVD.

Mitigations

Because attestation signing is a legitimate Microsoft program and the resulting drivers are signed with Microsoft certificates, execution-time detection is made much more difficult as most EDR tools and Anti-Viruses will allow binaries signed with Microsoft certificates to load. The recent blog post released by Mandiant on <u>UNC3944 driver operations</u> details multiple techniques that can be used by organizations to hunt for the abuse of attestation signing. If you haven't already, don't forget to read <u>part one on North Korea's UNC2970</u>. Additionally, Microsoft recently released a report detailing how organizations can <u>harden their environment against potentially vulnerable third-party developed drivers</u>.

Indicators of Compromise

MD5	Signature
def6f91614cb47888f03658b28a1bda6	XOR'd LIGHTSHIFT

9176f177bd88686c6beb29d8bb05f20c	LIGHTSHIFT
ad452d161782290ad5004b2c9497074f	LIGHTSHOW
7e6e2ed880c7ab115fca68136051f9ce	ENE Driver
SB_SMBUS_SDK.dll	LIGHTSHOW Dummy DLL
C:\Windows\windows.ini	LIGHTSHIFT Output

Signatures

LIGHTSHIFT

```
rule M_Code_LIGHTSHIFT
{
  meta:
        author = "Mandiant"
        description = "Hunting rule for LIGHTSHIFT"
       sha256 =
"ce501fd5c96223fb17d3fed0da310ea121ad83c463849059418639d211933aa4"
strings:
        $p00_0 = {488b7c24??448d40??48037c24??488bcfff15[4]817c24[5]74??
488b4b??33d2}
        $p00_1 = {498d7c01??8b47??85c075??496345??85c07e??8b0f41b9}
    condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and
        (
            ($p00_0 in (750..11000) and $p00_1 in (0..8200))
)
}
```

LIGHTSHOW

```
rule M_Code_LIGHTSHOW
{
meta:
        author = "Mandiant"
        description = "Hunting rule For LIGHTSHOW."
        md5 =
"ee5057da3e38b934dae15644c6eb24507fb5a187630c75725075b24a70065452"
    strings:
             $E01 = { 46 75 64 4d 6f 64 75 6c 65 2e 64 6c 6c }
             $I01 = { 62 63 72 79 70 74 2e 64 6c 6c }
             $I02 = { 4b 45 52 4e 45 4c 33 32 2e 64 6c 6c }
             $I03 = { 75 73 65 72 33 32 2e 64 6c 6c 00 }
        H1 = \{ 4D 5A 90 00 \}
         $H2 = { 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F }
             $F01 = { 47 65 74 4d 6f 64 75 6c 65 46 69 6c 65 4e 61 6d 65
57 }
             $F02 = { 47 65 74 4d 6f 64 75 6c 65 48 61 6e 64 6c 65 41 }
             $F03 = { 47 65 74 46 69 6c 65 54 79 70 65 }
             $F04 = { 47 65 74 56 65 72 73 69 6f 6e }
             $F05 = { 51 75 65 72 79 53 65 72 76 69 63 65 53 74 61 74 75
73 }
             $F06 = { 42 43 72 79 70 74 4f 70 65 6e 41 6c 67 6f 72 69 74
68 6d 50 72 6f 76 69 64 65 72 }
             $M01 = { 68 2d 79 6e b1 }
             $M02 = { 68 ea 71 c2 55 }
             M03 = \{ 66 b8 ad eb \}
             M04 = \{ 4c \ 8d \ 2c \ 6d \ b3 \ 6c \ 05 \ 39 \}
             M05 = \{ 48 \ 8d \ 2c \ 95 \ 08 \ 9d \ ec \ 9a \}
             S01 = \{ 48 \ 8d \ 0c \ f5 \ a3 \ cd \ 0a \ eb \}
             S02 = \{ 81 f9 7f 56 e6 0a \}
    condition:
```

(\$H1 in (0..2048)) and (\$H2 in (0..2048)) and filesize < 100MB and filesize > 5KB and all of (\$M0*) and all of (\$E*) and all of (\$I0*) and 6 of (\$F0*) and all of (\$S0*)

}