Analysis of APT28 hospitality malware (Part 2)

blog.xpnsec.com/apt28-hospitality-malware-part-2/

« Back to home

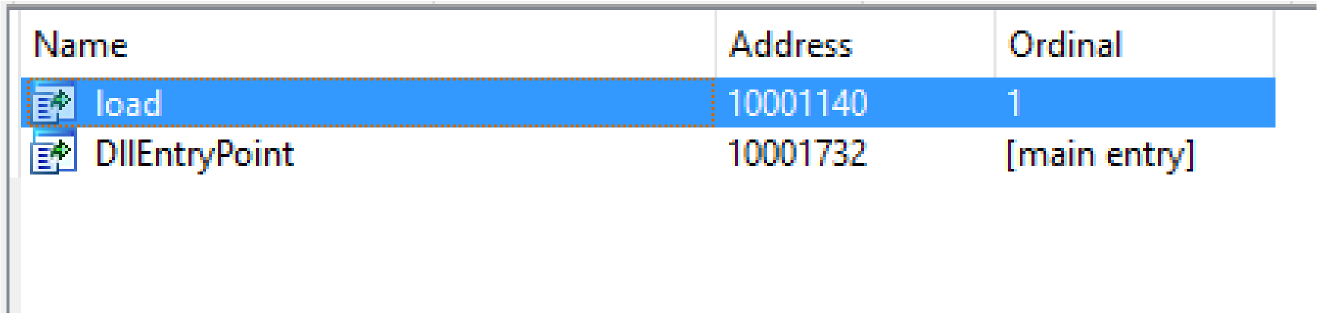
In the first part of this malware review, we looked at the VBA code used by APT28 to drop a DLL onto the victims' machine as part of their recently highlighted hospitality campaign.

In this post, we will look at the dropped file, and understand just what it does, and how we can analyse it using IDA Pro.

So we know from the first post that we have a DLL, which is run using the following command:

```
rundll32.exe %APPDATA%\user.dat,#1
```

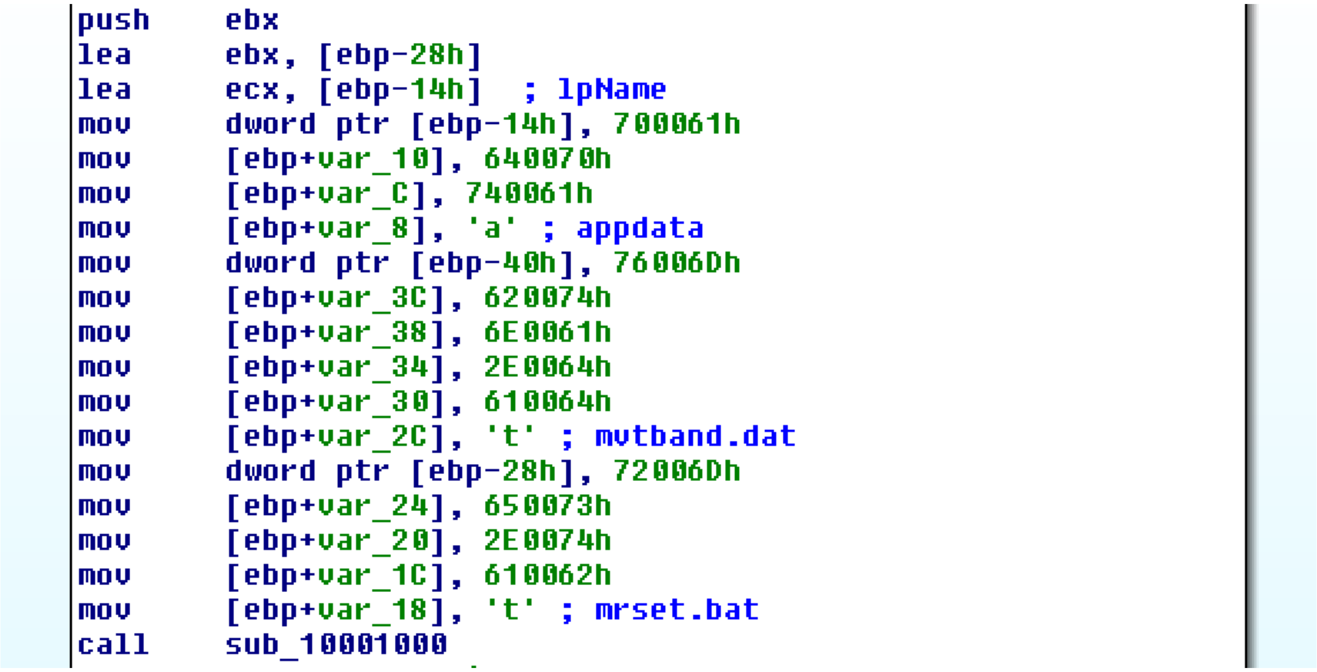
Loading the extracted DLL into IDA, the first thing that we notice is that we have an exported function of `load` with an ordinal of 1:



Name	Address	Ordinal
load	10001140	1
DllEntryPoint	10001732	[main entry]

We know from the rundll32.exe command that this will be our entry point, so we start our analysis here.

Within the `load` function, a number of strings are constructed on the stack in Unicode, which when decoded look like this:

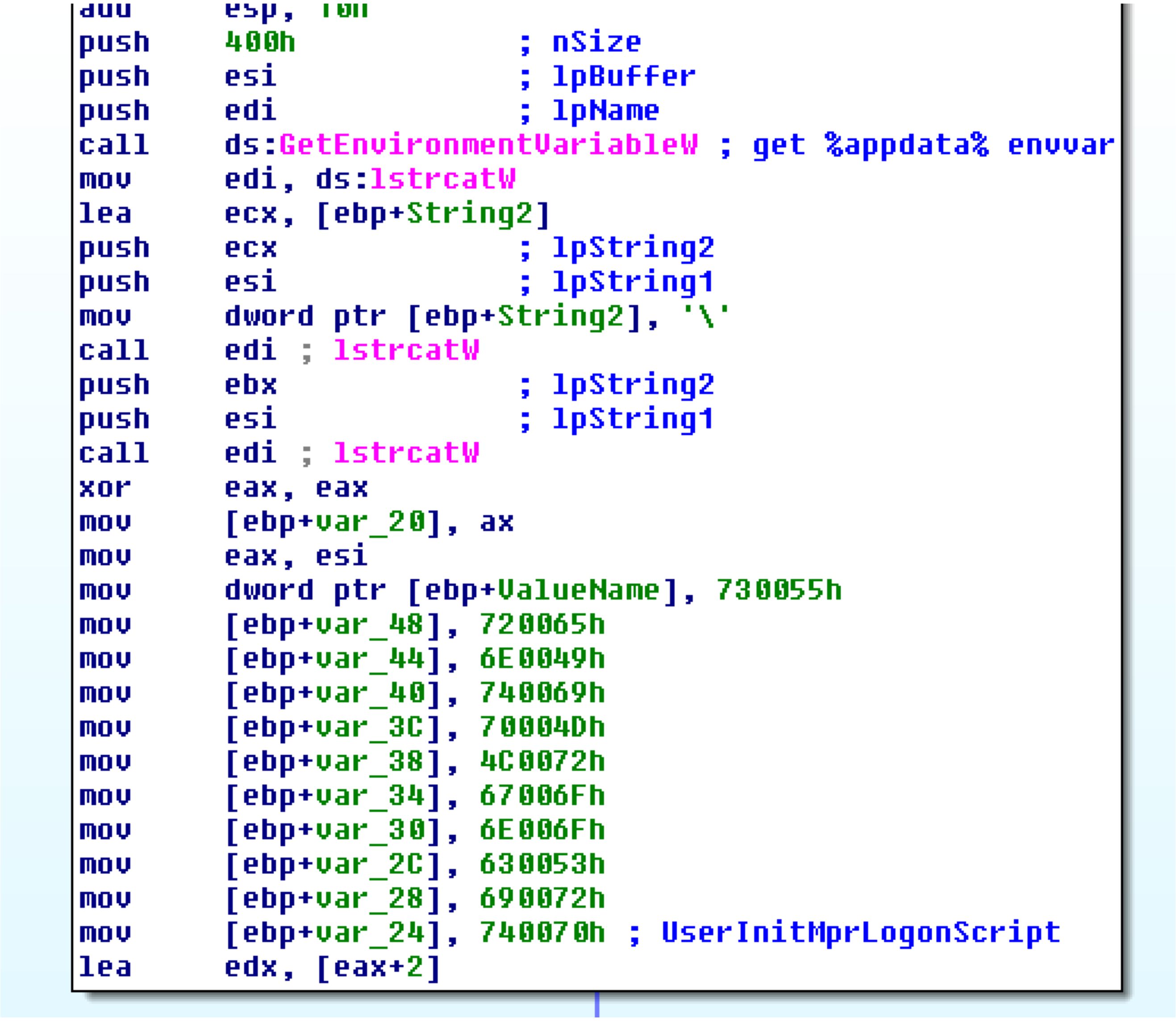


```
push    ebx
lea     ebx, [ebp-28h]
lea     ecx, [ebp-14h]  ; lpName
mov     dword ptr [ebp-14h], 700061h
mov     [ebp+var_10], 640070h
mov     [ebp+var_C], 740061h
mov     [ebp+var_8], 'a' ; appdata
mov     dword ptr [ebp-40h], 76006Dh
mov     [ebp+var_3C], 620074h
mov     [ebp+var_38], 6E0061h
mov     [ebp+var_34], 2E0064h
mov     [ebp+var_30], 610064h
mov     [ebp+var_2C], 't' ; mvtband.dat
mov     dword ptr [ebp-28h], 72006Dh
mov     [ebp+var_24], 650073h
mov     [ebp+var_20], 2E0074h
mov     [ebp+var_1C], 610062h
mov     [ebp+var_18], 't' ; mrset.bat
call    sub_10001000
```

Interestingly, one of the strings of `mvtband.dat` closely matches with the C2 server identified by FireEye of `mvtband.net`.

Entering the first function at address `10001000h`, we see another Unicode string constructed on the stack of “Environment” before `RegOpenKeyExW` is called to open a handle to `HKCU\Environment`.

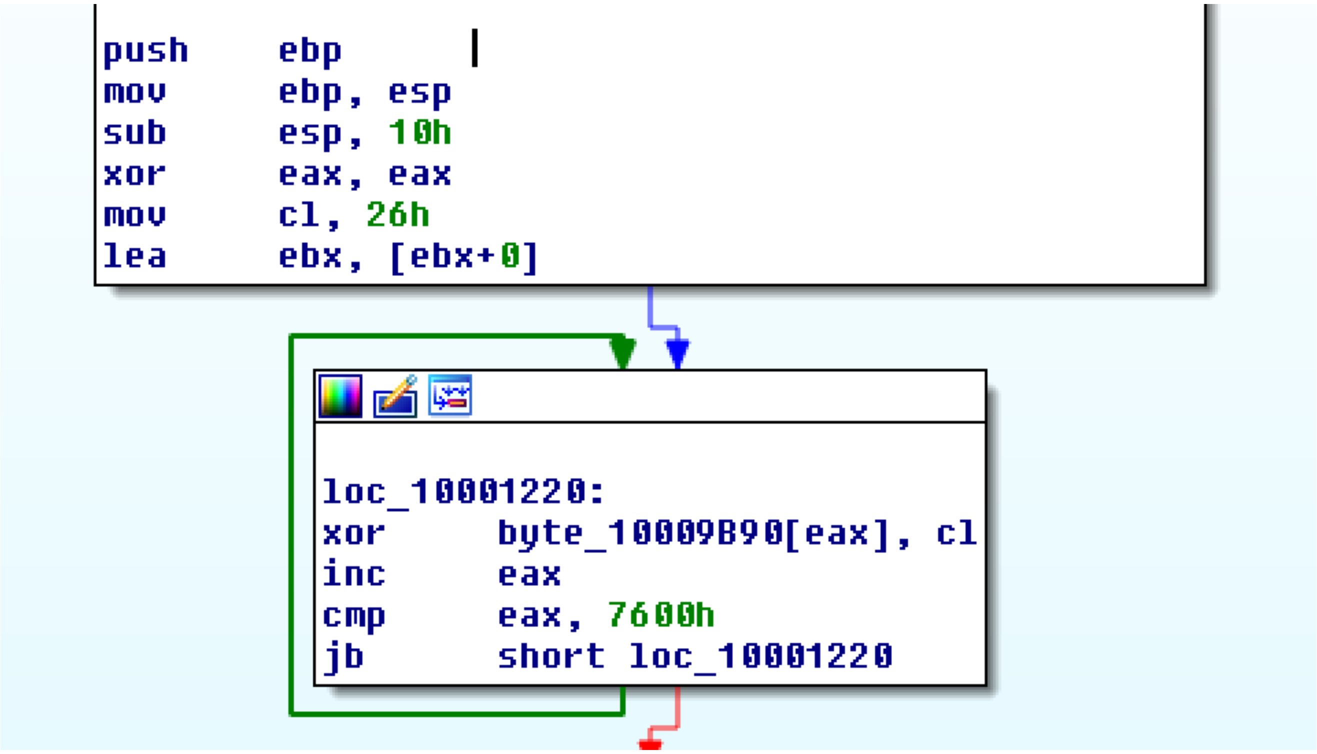
Next a path is constructed of `%appdata%\mrset.bat` and written to the `UserInitMprLogonScript` registry value within `HKCU\Environment`:



```
add     esp, 10h
push    400h            ; nSize
push    esi             ; lpBuffer
push    edi             ; lpName
call    ds:GetEnvironmentVariableW ; get %appdata% envvar
mov     edi, ds:lstrcatW
lea     ecx, [ebp+String2]
push    ecx             ; lpString2
push    esi             ; lpString1
mov     dword ptr [ebp+String2], '\'
call    edi ; lstrcatW
push    ebx             ; lpString2
push    esi             ; lpString1
call    edi ; lstrcatW
xor     eax, eax
mov     [ebp+var_20], ax
mov     eax, esi
mov     dword ptr [ebp+ValueName], 730055h
mov     [ebp+var_48], 720065h
mov     [ebp+var_44], 6E0049h
mov     [ebp+var_40], 740069h
mov     [ebp+var_3C], 70004Dh
mov     [ebp+var_38], 4C0072h
mov     [ebp+var_34], 67006Fh
mov     [ebp+var_30], 6E006Fh
mov     [ebp+var_2C], 630053h
mov     [ebp+var_28], 690072h
mov     [ebp+var_24], 740070h ; UserInitMprLogonScript
lea     edx, [eax+2]
```

Note: If we stop and look for other examples of malware using this technique, we can see a number of related posts unsurprisingly pointing to other Sofacy malware droppers using the same method.

Continuing to the next function, we find what immediately appears to be a decryption loop, using a fixed XOR key of 0x26:



```
push    ebp
mov     ebp, esp
sub     esp, 10h
xor     eax, eax
mov     cl, 26h
lea     ebx, [ebx+0]
```

```
loc_10001220:
xor     byte_10009B90[eax], cl
inc     eax
cmp     eax, 7600h
jb      short loc_10001220
```

One the bytes at this address are decrypted, the contents are written to `%appdata%\mvtband.dat`.

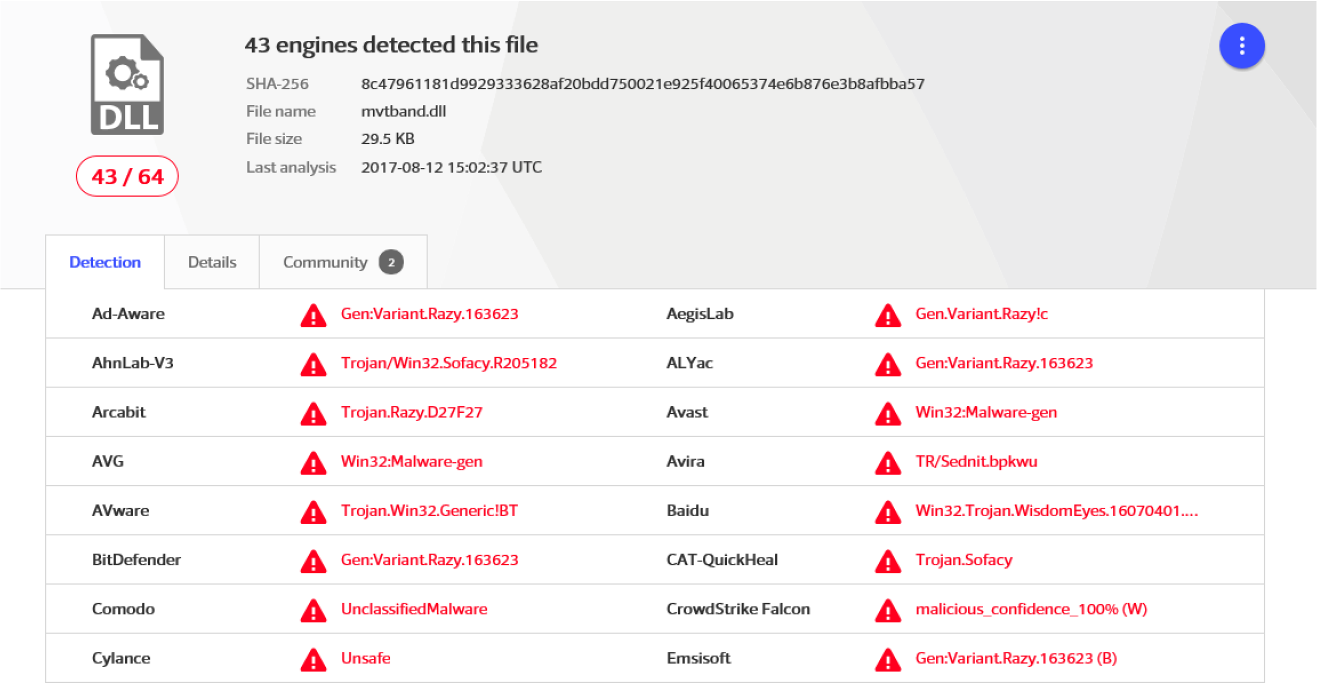
This is a perfect opportunity to use IDAPython to recover the encrypted data. We know from the disassembly that the loop runs for 0x7600 bytes, and XOR's a byte at a time from the address 0x10009B90 with a fixed key of 0x26. Translating this into IDAPython, we have the following script:

```
v = ""
bytes = idaapi.get_many_bytes(0x10009B90, 0x7600)
for i in range(0,len(bytes)):
    v += chr(ord(bytes[i]) ^ 0x26)

f = open("out.bin", "wb")
f.write(v)
f.close()
```

Once executed, this script will decrypt the contents of address `0x10009B90` and write the output to `out.bin`.

An initial review of the decrypted contents show that this is a PE32 DLL, and if we upload the sample to VirusTotal we see that a matching sample was first seen on 17-07-2017 with a name of `mvtband.dll` and signatures matching Sofacy:

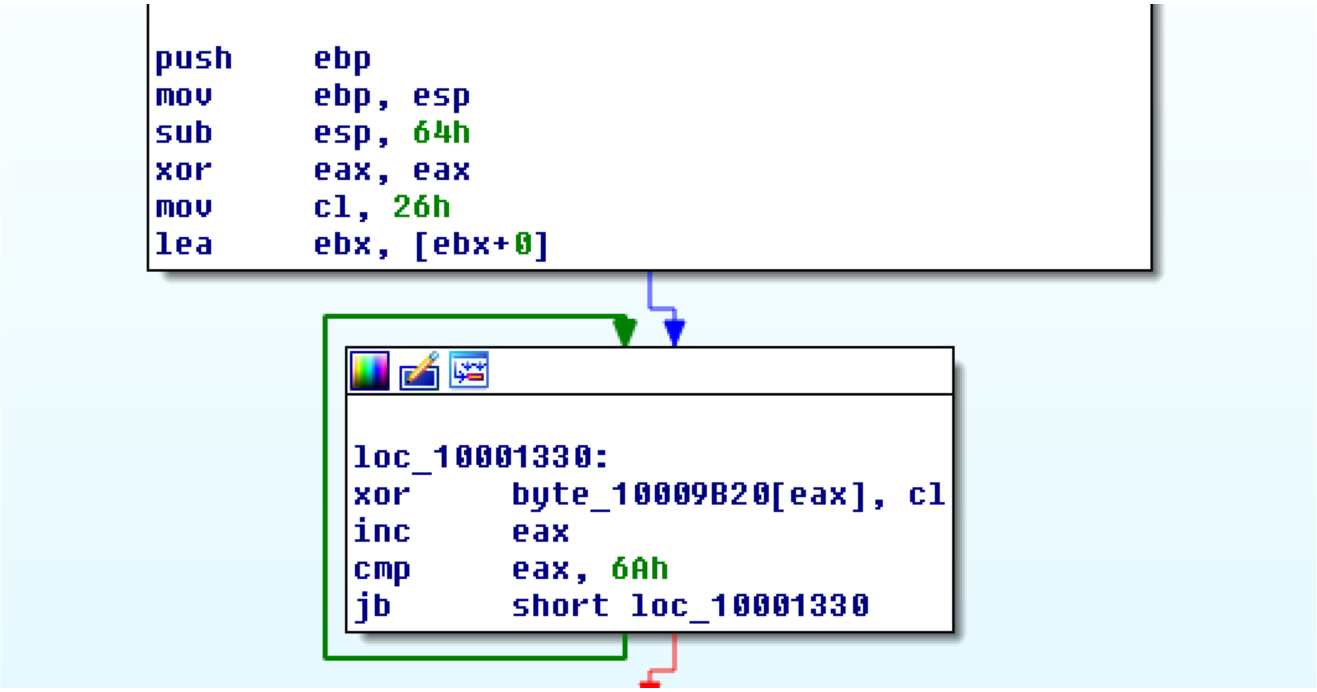


43 engines detected this file

SHA-256 8c47961181d9929333628af20bdd750021e925f40065374e6b876e3b8afbba57
File name mvtband.dll
File size 29.5 KB
Last analysis 2017-08-12 15:02:37 UTC

Ad-Aware	Gen:Variant.Razy.163623	AegisLab	Gen.Variant.Razy!c
AhnLab-V3	Trojan/Win32.Sofacy.R205182	ALYac	Gen:Variant.Razy.163623
Arcabit	Trojan.Razy.D27F27	Avast	Win32:Malware-gen
AVG	Win32:Malware-gen	Avira	TR/Sednit.bpkwu
AVware	Trojan.Win32.Generic!BT	Baidu	Win32.Trojan.WisdomEyes.16070401....
BitDefender	Gen:Variant.Razy.163623	CAT-QuickHeal	Trojan.Sofacy
Comodo	UnclassifiedMalware	CrowdStrike Falcon	malicious_confidence_100% (W)
Cylance	Unsafe	Emsisoft	Gen:Variant.Razy.163623 (B)

Continuing into the final function of this dropper, we find a similar decryption loop for a different memory location and the same XOR key:



```
push    ebp
mov     ebp, esp
sub     esp, 64h
xor     eax, eax
mov     cl, 26h
lea     ebx, [ebx+0]
```

```
loc_10001330:
xor     byte_10009B20[eax], cl
inc     eax
cmp     eax, 6Ah
jb      short loc_10001330
```

Repurposing our above IDAPython script, we can extract the contents with the following:

```
v = ""
bytes = idaapi.get_many_bytes(0x10009B20, 0x6A)
for i in range(0,len(bytes)):
    v += chr(ord(bytes[i]) ^ 0x26)

f = open("out2.bin", "wb")
f.write(v)
f.close()
```

Reviewing the decrypted contents, we find the following:

```
set inst_pck = "%appdata%\mvtband.dat"
if NOT exist %inst_pck % (exit)
start rundll32.exe %inst_pck %,#1
```

This simple .bat file is being used by the `UserInitMprLogonScript` registry value on reboot to launch the `mvtband.dat` payload via rundll32.exe.

Once the .bat file script is decrypted by the dropper, the contents are written to `%appdata%\mrset.bat` before being launched using `CreateProcess`.

And there we have it, APT28's simple dropper and persistence malware, with a bit of IDAPython reversing thrown in. We see that this DLL functions to decrypt 2 embedded payloads, "mrset.bat" which is a BAT file executed by "UserInitMprLogonScript", and "mvtband.dat" which is the main payload of the malware which is executed via rundll32.exe.

So what are the takeaways from this for our red-team engagements? Well first, we see that adversaries are now increasingly using rundll32.exe in malware campaigns, which allows a payload to be stored without a typical .exe extension. More importantly, this also gives malware a better chance at being successfully executed within a restricted environment which whitelists Microsoft signed binaries.

Secondly, we have `UserInitMprLogonScript` being used for persistence to launch a .bat file as a GPO script. While certainly not unheard of, the use of a GPO value is less likely to draw attention than say, adding a RUN key value, or adding a new schtask.

Hopefully this has been a good introduction to the APT28 dropper and how we can use IDAPython during a reversing exercise, and as always, comments and feedback are welcome via the usual channels.