Mustang Panda Recent Activity: Dll-Sideloading trojans with temporal C2 servers

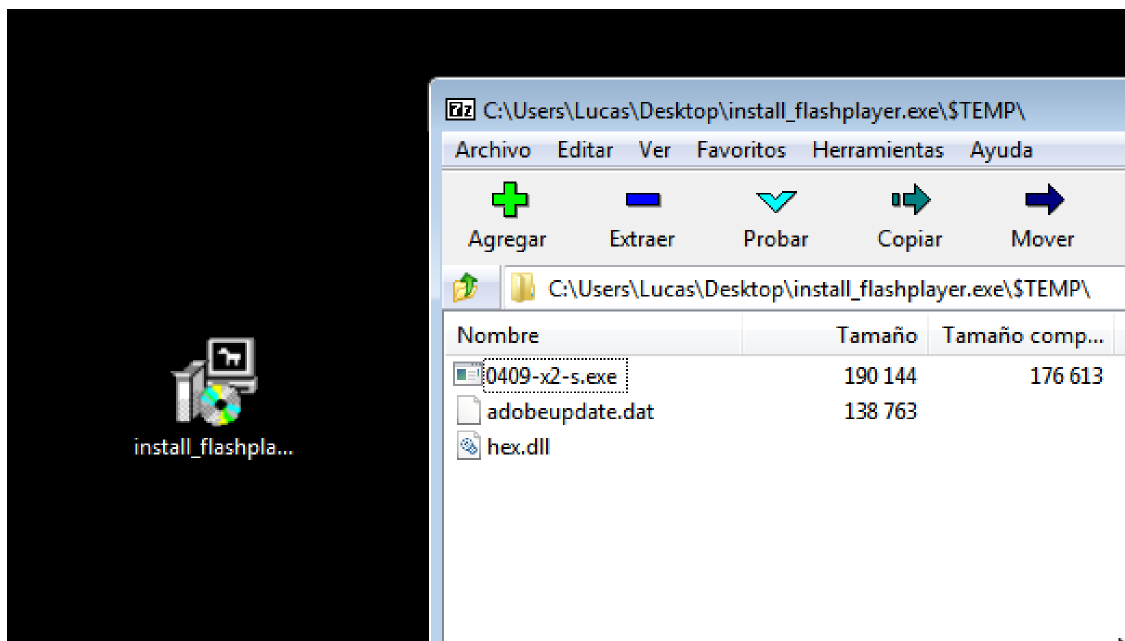
Published: 2020-06-02 · Archived: 2026-04-05 18:50:37 UTC

Recently, from Lab52 we have detected a recent malware sample, using the Dll-Sideload technique with a legitimate binary, to load a threat.

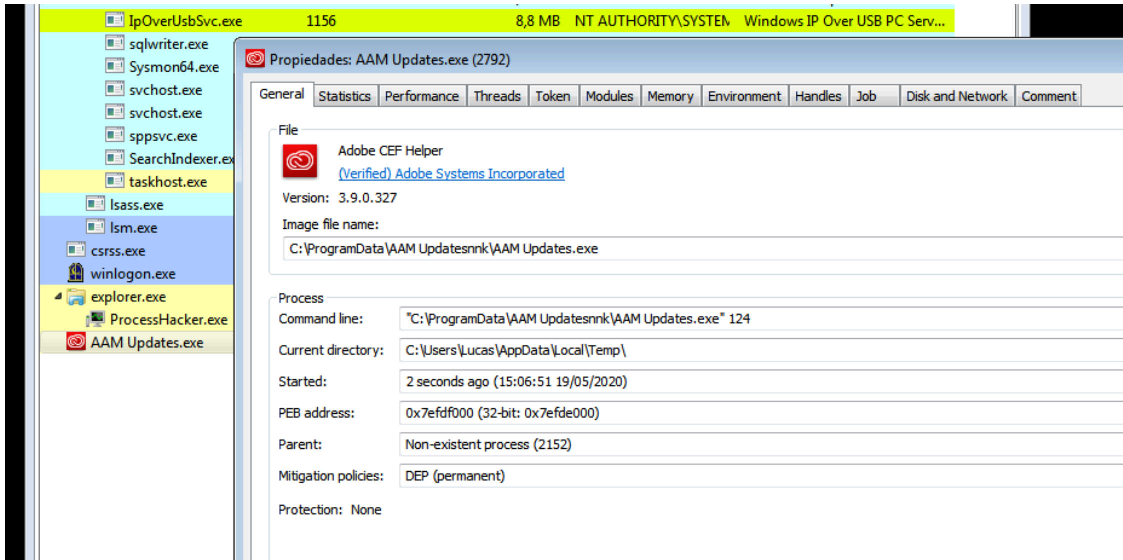
This particular sample has a very small DLL, that loads an encrypted file, which after being decrypted consists of a sample of the PlugX Trojan. This technique, and final threat together, consists of one of the most common TTPs among some APT groups generally of Chinese origin such as APT1, APT27 and Mustang Panda.

The sample in question is downloaded from the following link "http://miandfish.]store/player/install_flash_player.exe" and although in previous months, it had another hash, currently the sample hosted under that name has the following hash "c56ac01b3af452fedc0447d9e0fe184d093d3fd3c6631aa8182c752463de570c".

The binary consists of an installer, which drops in the folder "C:\ProgramData\AAM Updatesnnk" the legitimate binary vulnerable to dll sideload, the small dll that acts as a loader for the final threat, and the binary file, which consists of the encrypted PlugX sample.



After deploying the three files, the installer runs the legitimate binary, causing the final PlugX threat to be loaded by it.



In this case, the legitimate vulnerable binary is part of Adobe's Swite which will load any library named "hex.dll" that is next to the executable.

Signature Verification

Signed file, valid signature

File Version Information

Copyright	Copyright 2013-2016 Adobe Systems Incorporated. All rights reserved.
Product	Adobe CEF Helper
Description	Adobe CEF Helper
Original Name	Adobe CEF Helper.exe
Internal Name	Adobe CEF Helper.exe
File Version	3.9.0.327
Date signed	1:22 AM 10/13/2016

Signers

+ Adobe Systems Incorporated
+ Symantec Class 3 Extended Validation Code Signing CA - G2
+ VeriSign

That hex.dll, in this case is a very simple and relatively small loader:

Function name	Segment
sub_10001000	.text
nreoiyoaynioytrupeyfk	.text
oojqwhnfjutcrejlbnxvds	.text
dqysrefqdhwpbgfudfbhlsqxrhrdpwu	.text
lujmewohkm	.text
DecodeFile	.text
OtroReadFile	.text
Open_file	.text
CEFProcessForkHandlerEx	.text
DllMain(x,x,x)	**.text**
operator delete(void *)	.text
memset	**.text**
operator new(uint)	.text
exit	**.text**
strncat	**.text**
strncpy	**.text**
strlen	**.text**
strrchr	**.text**
fclose	**.text**
ftell	**.text**
fseek	**.text**
fopen	**.text**
_CRT_INIT(x,x,x)	.text

It has 4 exports that return 0 without doing anything, the Main function of the library, on the other hand, calls a function that checks the existence of the .dat file which is hardcoded (adobeupdate.dat in this case), loads it, extracts the first string of the binary and uses it as XOR key to decode the rest of the file, which consists on the final threat.

The following code in python imitates the logic of decoding:

```
import sys

args = sys.argv[1:]

for file in args:
    #Read xored file
    f = open(file,"r")
    Data=f.read()
    f.close()
    DataLenth=len(Data)

    #Get first string (until caracter 0x00...)
    XorKey = Data.split(chr(0x0))[0]
    XorKeyLength = len(XorKey)

    res = ""

    #Decode the rest of the file with the xor key
    for i in range(DataLenth-XorKeyLength-1):
        res+= chr(ord(Data[i+XorKeyLength+1])^ord(XorKey[i%XorKeyLength]))

    #Save the decoded file
    fr = open("plugx.exe","w")
    fr.write(res)
    fr.close()
```

When it finishes deciphering it, it loads the malware into memory, makes a “Memprotect” to make it executable and launches its logic from the byte 0 of the binary.

It is a functional PE, so this should not work, since it starts with the “MZ” header of a normal binary:

```
4D 5A E8 00 00 00 00 5B 52 45 55 8B EC 81 C3 69  MZè....[REU‹ì.Ãi
13 00 00 FF D3 C9 C3 00 40 00 00 00 00 00 00 00  ...ÿÓÉÃ.@.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00  ............ø...
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
```

But in this case it uses a technique already seen before in tools like the Cobalt Strike Beacon that by modifying some bytes of the MZ header, it becomes meaningful executable code.

If we open the binary as a shellcode (de-compiling from byte 0) we see how they have modified the first bytes into a routine that jumps to a code zone, consisting of a PE loader:

```
seg000:00000000                    sub_0     proc near        ; DATA XREF: sub_12210+5↓r
seg000:00000000                                               ; sub_12210+3E↓w ...
seg000:00000000 4D                           dec     ebp
seg000:00000001 5A                           pop     edx
seg000:00000002 E8 00 00 00 00               call    $+5
seg000:00000007 5B                           pop     ebx
seg000:00000008 52                           push    edx
seg000:00000009 45                           inc     ebp
seg000:0000000A 55                           push    ebp
seg000:0000000B 8B EC                        mov     ebp, esp
seg000:0000000D 81 C3 69 13 00 00            add     ebx, 1369h
seg000:00000013 FF D3                        call    ebx
seg000:00000015 C9                           leave
seg000:00000016 C3                           retn
seg000:00000016                    sub_0     endp
```

After loading the IAT and leaving everything ready as a normal executable, this threat decrypts its own config, which is encrypted in XOR in the .data section of the binary. This time the decryption key is hardcoded in the binary, and is the string “123456789”.

After decrypting its configuration, it contains the folder where the binary must be installed, a XOR key that will use to encrypt it’s traffic and a list of up to 4 domains or IP addresses of command and control servers together with the port to be used. Generally the 4 C2 elements consists of the same domain repeated 4 times or 2 domains repeated twice each.

After the analysis, both the loader in DLL format and the final encrypted threat (after decryption) have been compared with different campaign samples of groups known to use this dll sideload technique, and it has been possible to verify how both the loader and the final threat coincide in a high percentage with the samples of the “Mustang Panda” group analyzed in the following reports [1] [2] [3]. In fact, the loader of this campaign is able to load and run the samples of the campaigns analyzed in those reports, and the final threat uses exactly the same XOR key to decipher its configuration as the samples in those reports, so there is a high probability that it is a new campaign from this same group.

This particular sample has the domains “www.destroy2013.]com” and “www.fitehook.]com” as c2 servers, and we have seen that they have a very characteristic behavior, since most of the day they resolve to 127.0.0.1, but from

1-3 AM (UTC) to 8-9 AM (UTC) it resolves to the IP “107.150.112.]250, except for weekends that it resolves constantly to 127.0.0.1, which could indicate that it is a campaign that is focused on a time zone in which those hours are working hours.

IP	81.16.28.]30
IP	107.150.112.]250
DOMAIN	www.destroy2013.]com
DOMAIN	www.fitehook.]com
DOMAIN	miandfish.]store
SHA256	c56ac01b3af452fedc0447d9e0fe184d093d3fd3c6631aa8182c752463de570c
SHA256	9c0f6f54e5ab9a86955f1a4beffd6f57c553e34b548a9d93f4207e6a7a6c8135

Source: https://lab52.io/blog/mustang-panda-recent-activity-dll-sideloading-trojans-with-temporal-c2-servers/