Let's Learn: Dissecting Operation ShadowHammer Shellcode Internals in crt_ExitProcess

vkremez.com/2019/03/lets-learn-dissecting-operation.html

Goal: Reverse engineer and document the Operation ShadowHammer malware and its shellcode in-depth as it was originally discovered and reported by Kaspersky Labs.
Source:

> Exclusive: Kaspersky researchers uncover operation #ShadowHammer - a supplychain attack targeting high-profile users. #KLResearch https://t.co/uyjK3lHP95 — Securelist (@Securelist) March 25, 2019

Sample:
Original Sample -> Setup.exe (MD5: 55a7aa5f0e52ba4d78c145811c830107)
Outline:

```
I. Background & Summary
II. Main Shellcode Decoder & Execution Function
III. GetAdaptersInfo Media Access Control (MAC) Parser
IV. Compare MAC Computed MD5 Hash Against Hardcoded MD5 Set
V. Call Command-and-Control Server Function
VI. INI File Logger Function
Appendix: Hardcoded Targeted MAC Addresses
```

I. Background & Summary: Operation ShadowHammer is a newly discovered supply chain attack that leveraged ASUS Live Update software, originally discovered and reported by Kaspersky Labs. This case is interesting as it involves a lengthy targeting via the compromised supply chain attack vector. The attackers deploy an interesting shellcode within the backdoored ASUS "Setup.exe," which is stored within “crt_ExitProcess” function. The main malware function contains a unique decoder blob with constants/XOR function that is responsible for decoding and executing its core shellcode. By and large, the shellcode performs MAC lookup logic for victims of interest via parsing GetAdaptersInfo API return and struct AdapterAddresses for physical devices media access control (MAC) numbers, which are oftentimes unique enough to identify PCs. Moreover, the malware shellcode deployed MD5Init, MD5Update, and MD5Final API calls to create an MD5 hash of the MAC address treated as byte object. The shellcode leverages the WININET library to call the server appending "?"+ unique identifier if there is a match with the identified hardcoded MD5 value and the victim machine MAC MD5. Otherwise, the shellcode is responsible for writing the file to the local user directory while altering the timestamps in the written file leveraging FileTimeToSystemTime and adding 0x93A80 => 604800 seconds => 7 days. It is, currently, one of the oddities of this malware. The purpose of this date alteration ahead is unclear at

this time. The original MISP JSON, CSV, and STIX indicators of compromise (IOCs) as well as the raw extracted and decoded shellcode is available here on my GitHub. You can simply check if you are targeted by the MD5 MAC match by generating an MD5 value of physical device MAC addresses and comparing against the hardcoded MD5 ones from this malware sample as provided below.

II. Main Shellcode Decoder & Execution Function

The attackers deploy the shellcode decoder and execution routine within the ASUS "Setup.exe" as seen below.

The "shelldecoder" function allocates the memory for the obfuscated shellcode, decodes it, and executes it.
The decoding algorithm employed is allocated and written in two steps decoding the first 16 length blob of the encoded payload and then the rest with the same decoding algorithm:

The main actual decoder algorithm function is peculiar and as follows (h/t @xorsthingsv2):

```
int __stdcall decoder_func(int *a1, int a2, int a3)
{

...

  memset(&v4, 0xCCu, 0x108u);
  v10 = 0;
  v9 = 0;
  v8 = *a1;
  v7 = *a1;
  v6 = *a1;
  v5 = *a1;
  do
  {
    v8 = v8 + (v8 >> 3) - 0x11111111;
    v7 = v7 + (v7 >> 5) - 0x22222222;
    v6 += 0x33333333 - (v6 << 7);
    v5 += 0x44444444 - (v5 << 9);
    v9 = v5;
    *(_BYTE *)(v10 + a3) = (v5 + v6 + v7 + v8) ^ *((_BYTE *)a1 + v10);
    result = ++v10;
  }
  while ( v10 < a2 );
  return result;
}
```

Next, the shellcode resolves a plethora of API calls via process environment block (PEB) traversal technique.

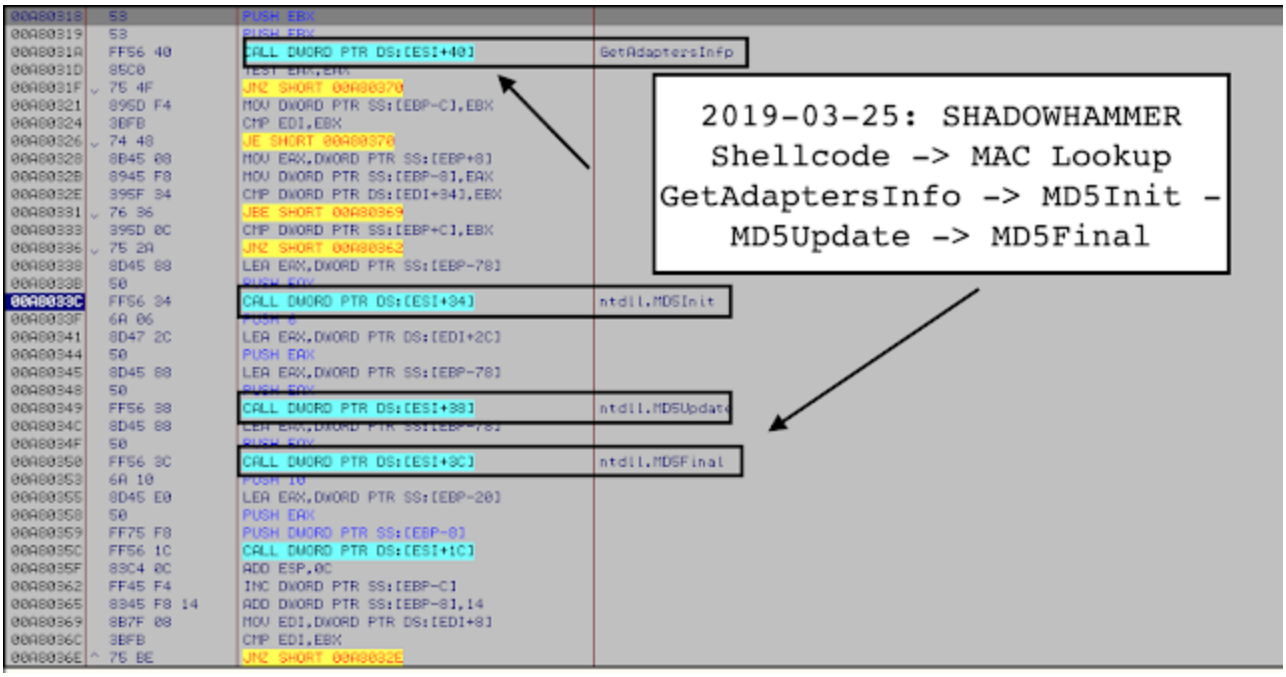
```
////////////////////////////////////////////////////
////////// OP ShadowHammer Resolved APIs ///////////////////
////////////////////////////////////////////////////
0028FC48   76E250C1  kernel32.LoadLibraryExW
0028FC4C   76E2C43A  kernel32.VirtualAlloc
0028FC50   76E2EF35  kernel32.GetModuleFileNameW
0028FC54   76E18649  kernel32.WritePrivateProfileStringW
0028FC58   76E2D816  kernel32.GetSystemTimeAsFileTime
0028FC5C   76E2BE0D  kernel32.FileTimeToSystemTime
0028FC60   76E36B15  kernel32.VirtualFree
0028FC64   770A4CC0  ntdll.memcpy
0028FC68   770A3B2F  ntdll.memcmp
0028FC6C   770A5340  ntdll.memset
0028FC70   7713A569  ntdll.swprintf
0028FC74   7713A41F  ntdll.sprintf
0028FC78   770A5640  ntdll.strncat
0028FC7C   77079DDC  ntdll.MD5Init
0028FC80   77079EA4  ntdll.MD5Update
0028FC84   77079E16  ntdll.MD5Final
0028FC88   72B36A4D  IPHLPAPI.GetAdaptersAddresses
0028FC8C   7618B880  WININET.InternetOpenA
0028FC90   76246000  WININET.InternetOpenUrlA
0028FC94   76187540  WININET.InternetQueryDataAvailable
0028FC98   76181C80  WININET.InternetReadFile
```

Then, the shellcode enters its main function responsible for obtaining and comparing MAC addresses and calling the server, among other functionality:

```
//////////////////////////////////////////////////
////////// OP ShadowHammer Main Function /////////////////
//////////////////////////////////////////////////
....
  result = GetAdaptersInfo_Alloc(a1, 0, 1);
  if ( result > 0 )
  {
    v3 = 20 * (result + 5);
    hash_mac_alloc = VirtualAlloc(
                       0,
                       20 * (result + 5),
                       12288,
                       4);
    memset(hash_mac_alloc, 0, v3);
    result = GetAdaptersInfo_Alloc(v1, hash_mac_alloc, 0);
    v4 = result;
    if ( result > 0 )
    {
       memset(&v5, 0, 44);
      if ( qmempy_memcmp_for_computed_md5_mac(v1, &v6, hash_mac_alloc, v4, (int)&v5)
)
        result = call_c2(v1, (int)&v5);
      else
        result = file_creation_log(v1);
    }
  }
  return result;
```

III. GetAdaptersInfo Media Access Control (MAC) Parser

One of the most notable functions of the malware simply parses GetAdaptersInfo API return and struct AdapterAddresses for physical devices media access control (MAC) numbers via EDI+2C return, which is used to generate an MD5 hash value of the length 16 as seen below.



The shortened pseudocoded function is as follows:

hashta< MISP JSON/CSV/#STIX hashtawith the extracted raw shellcode to https://lnkd.in/dMcZJ_g

```
//////////////////////////////////////////////////
/////////// OP ShadowHammer Obtain & Hash MAC ////////////////
//////////////////////////////////////////////////
 int __usercall GetAdaptersInfo_Alloc@<eax>(int a1@<esi>, int hash_match, int a3)
{

....
  v9 = 0;
  if ( GetAdaptersInfo (0, 0, 0, 0, &v9) == 0x6F )// 0x6F BUFFER_RUN_OVERFLOW
  {
    VirtualAlloc_ret = VirtualAlloc(0, v9, 4096, 4);
    if ( !GetAdaptersInfo(0, 0, 0, VirtualAlloc_ret, &v9) )

      v7 = 0;
      if ( VirtualAlloc_ret )
      {
        dest = hash_match;
        do
        {
          if ( *(_DWORD *)(VirtualAlloc_ret + 52) > 0u )
          {
            if ( !a3 )
            {
              MD5Init(&MD5_CTX_context);
              MD5Update(&MD5_CTX_context, VirtualAlloc_ret + 44, 6);// len = 6
              MD5Final(&MD5_CTX_context);
              memcpy(dest, &source, 16);
            }
            ++v7;
            dest += 20;
          }
          VirtualAlloc_ret = *(_DWORD *)(VirtualAlloc_ret + 8);
        }
        while ( VirtualAlloc_ret );
      }
    }
    result = v7;
  }
  else
  {
    result = 0;
  }
  return result;
```

IV. Compare MAC Computed MD5 Hash Against Hardcoded MD5 Set

The shellcode simply checks via memcmp API and parses the list of the hardcoded MD5 values against the machine generates MD5 MAC addresses via the following excerpt:

```
//////////////////////////////////////////////////////
///// OP ShadowHammer Compare MD5 Hash Excrept ////////
//////////////////////////////////////////////////////
    while ( 1 )
  {
    v18 = *(_DWORD *)v20;
    if ( v18 == 1 )
    {
      qmemcpy(&v13, v20, 0x2Cu);
      v5 = 0;
      v19 = 0;
      if ( a4 )
      {
        mac_machine = a3;
        while ( memcmp(&pre_computed, mac_machine, 16) )
        {
          ++v19;
          mac_machine += 20;
          if ( v19 >= a4 )
            goto LABEL_9;
        }
        v5 = 1;
      }
```

The matched excerpt list of the hardcoded MD5 hashes is as follows:

```
//////////////////////////////////////////////////////
///// OP ShadowHammer Hardcoded MD5 List //////////////
//////////////////////////////////////////////////////

00B006C7DAB6ACE6C25C3799EB2B6E14
5977BAA3F8CE0CA1C96D6AC9A40C9A91
409D8EEBCE8546E56A0AD740667AADBD
7DA42DD34574D4E1A7EA0E708E7BC9A6
ADE62A257ADF118418C5B2913267543E
4268AED64AA5FFF2020D2447790D7D32
7B14C53FD3604CC1EBCA5AF4415AFED5
3A8EA62E32B4ECBE33DF500A28EBC873
CC16956C9506CD2BB389A7D7DA2433BD
FE4CCC64159253A6019304F17102886D
F241C3073A5777742C341472E2D43EEC
AB0CEF9E5957129E23FBA178120FA20B
F758024E734077C70532E90251C5DF02
F35A60617AB336DE4DAAC799676D07B6
6A62EAD801802A5C9EC828D0C1EDBB5B
600C7B52E7F80832E3CEE84FCEC88B9D
6E75B2D7470E9864D19E48CB360CAF64
FB559BCD103EE0FCB0CF4161B0FAFB19
690AD61EC7859A0964216B66B5D33B1A
09DA9DF3A050AFAD0DF0EF963B41B6E2
FAE3B06AB27F2B0F7C29BF7F2B03F83F
D4B958671F47BF5DCD08705D80DE9A53
```

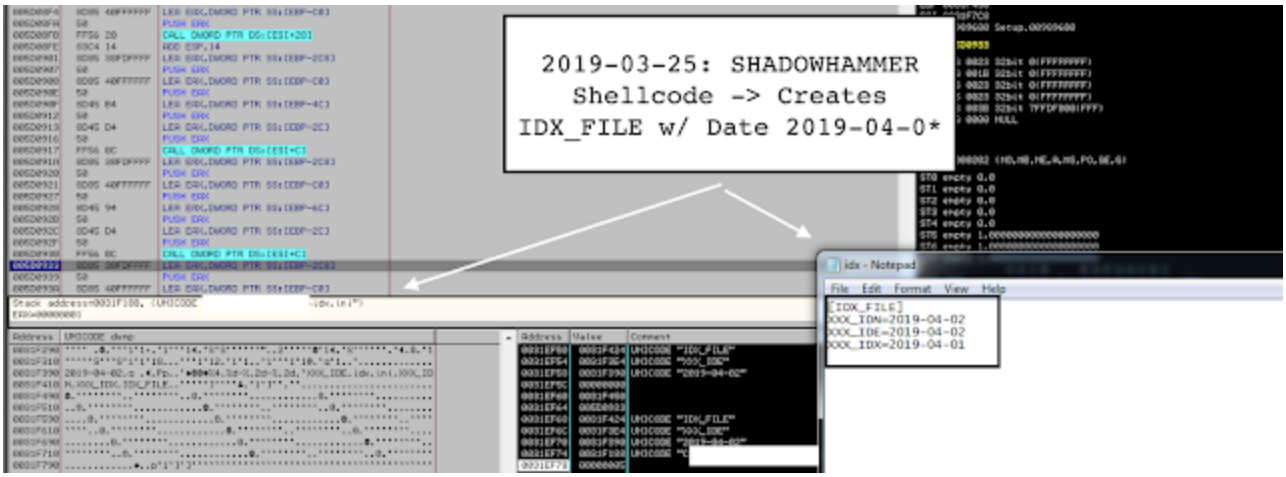
V. Call Command-and-Control (C2) Server Function

If the malware is able to match successfully the hardcoded MD5 values and the machine MAC MD5 address, it proceeds to the C2 call function; otherwise, it writes a .ini file. The C2 server call function is rather trivial relying on the WININET DLL and the following API calls ANSI-version InternetOpenA, InternetOpenUrlA, InternetReadFile, InternetQueryDataAvailable.

The call appends and formats via sprintf API the URL to "?" and the unique identifier of the victim added to the URL string hxxps://asushotfix[.]com/logo2[.]jpg.

VI. INI File Logger Function When the malware fails to find a match with the hardcoded MD5 MAC addresses, it calls this interesting .ini logger function essentially creating a file called "idx.ini" in directory with the timestamp set to +7 days from the current system timestamp. 2019-03-27 update (h/t @hFireF0X and @KyleHanslovan): the shellcode leverages GetModuleFileNameW to drop the .ini file two directories deep from the initial execution path. For example, if the path is C:\Users\USER\Desktop, it will drop the file to C:\Users\, if it is C:\Program Files\ASUS, it will drop the file to C:\.

The shellcode also sets up the time +7 days as it is one of the oddities of the malware.



The pseudo-coded function is as follows:

```
//////////////////////////////////////////////////////
///// OP ShadowHammer Time Alter Function //////////
//////////////////////////////////////////////////////

    memset((__int16 *)&v10, 0, 16);
    GetSystemTimeAsFileFileTime(&v67);// GetSystemTimeAsFileFileTime
    v3 = time_proc(v67 - 0x19DB1DED53E8000i64, 0x989680u, 0);
    if ( v3 > 32535244799i64 )
    {
      LODWORD(v3) = 0xFFFFFFFF;
      HIDWORD(v67) = 0xFFFFFFFF;
    }
    v4 = (signed int)v3 + 0x93A80 - 0x49EF6F00i64;// 0x93A80 = 604800 seconds = 7
days
    HIDWORD(v5) = HIDWORD(v4) + 2;
    LODWORD(v5) = v4;
    v68 = time_proc(v5, 0x989680i64);
    FileTimeToSystemTime(&v68, (__int16 *)&v10);
    swnprintf(&v9, &v13, v10, v11, v12);
    WritePrivateProfileStringW(&v58, &v42, &v9, (__int16 *)&v8);
    WritePrivateProfileStringW(&v58, &v26, &v9, &v8);
    result = WritePrivateProfileStringW(&v58, &v50, &v9, (__int16 *)&v8);
  }
  return result;
}
```

Appendix: Hardcoded Targeted MAC Addresses

```
//////////////////////////////////////////////////////
///// OP ShadowHammer Hardcoded MD5 List //////////////
//////////////////////////////////////////////////////

00B006C7DAB6ACE6C25C3799EB2B6E14
5977BAA3F8CE0CA1C96D6AC9A40C9A91
409D8EEBCE8546E56A0AD740667AADBD
7DA42DD34574D4E1A7EA0E708E7BC9A6
ADE62A257ADF118418C5B2913267543E
4268AED64AA5FFF2020D2447790D7D32
7B14C53FD3604CC1EBCA5AF4415AFED5
3A8EA62E32B4ECBE33DF500A28EBC873
CC16956C9506CD2BB389A7D7DA2433BD
FE4CCC64159253A6019304F17102886D
F241C3073A5777742C341472E2D43EEC
AB0CEF9E5957129E23FBA178120FA20B
F758024E734077C70532E90251C5DF02
F35A60617AB336DE4DAAC799676D07B6
6A62EAD801802A5C9EC828D0C1EDBB5B
600C7B52E7F80832E3CEE84FCEC88B9D
6E75B2D7470E9864D19E48CB360CAF64
FB559BCD103EE0FCB0CF4161B0FAFB19
690AD61EC7859A0964216B66B5D33B1A
09DA9DF3A050AFAD0DF0EF963B41B6E2
FAE3B06AB27F2B0F7C29BF7F2B03F83F
D4B958671F47BF5DCD08705D80DE9A53
```

V. Call Command-and-Control (C2)