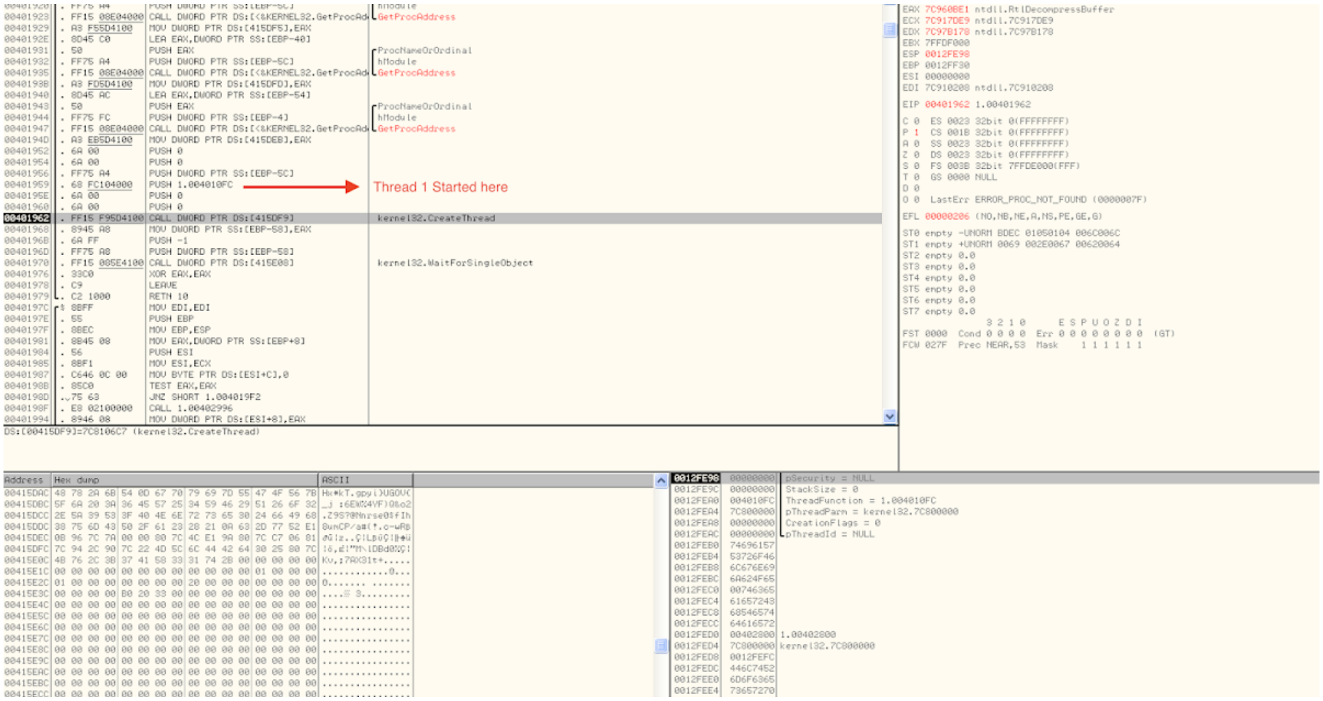
Unpacking Payload used in Bottle EK

pwncode.io/2019/12/unpacking-payload-used-in-bottle-ek.html



```
00401920 . FF75 A4        PUSH DWORD PTR SS:[EBP-5C]                       hModule
00401923 . FF15 08E04000  CALL DWORD PTR DS:[<&KERNEL32.GetProcAd          GetProcAddress
00401929 . A3 F55D4100    MOV DWORD PTR DS:[415DF5],EAX
0040192E . 8D45 C0        LEA EAX,DWORD PTR SS:[EBP-40]
00401931 . 50             PUSH EAX                                         ProcNameOrOrdinal
00401932 . FF75 A4        PUSH DWORD PTR SS:[EBP-5C]                       hModule
00401935 . FF15 08E04000  CALL DWORD PTR DS:[<&KERNEL32.GetProcAd          GetProcAddress
0040193B . A3 FD5D4100    MOV DWORD PTR DS:[415DFD],EAX
00401940 . 8D45 AC        LEA EAX,DWORD PTR SS:[EBP-54]
00401943 . 50             PUSH EAX                                         ProcNameOrOrdinal
00401944 . FF75 FC        PUSH DWORD PTR SS:[EBP-4]                        hModule
00401947 . FF15 08E04000  CALL DWORD PTR DS:[<&KERNEL32.GetProcAd          GetProcAddress
0040194D . A3 EB5D4100    MOV DWORD PTR DS:[415DEB],EAX
00401952 . 6A 00          PUSH 0
00401954 . 6A 00          PUSH 0
00401956 . FF75 A4        PUSH DWORD PTR SS:[EBP-5C]
00401959 . 68 FC104000    PUSH 1.004010FC               ------->          Thread 1 Started here
0040195E . 6A 00          PUSH 0
00401960 . 6A 00          PUSH 0
00401962 . FF15 F95D4100  CALL DWORD PTR DS:[415DF9]                       kernel32.CreateThread
00401968 . 8945 A8        MOV DWORD PTR SS:[EBP-58],EAX
0040196B . 6A FF          PUSH -1
0040196D . FF75 A8        PUSH DWORD PTR SS:[EBP-58]
00401970 . FF15 085E4100  CALL DWORD PTR DS:[415E08]                       kernel32.WaitForSingleObject
00401976 . 33C0           XOR EAX,EAX
00401978 . C9             LEAVE
00401979 . C2 1000        RETN 10
0040197C $ 8BFF           MOV EDI,EDI
0040197E . 55             PUSH EBP
0040197F . 8BEC           MOV EBP,ESP
00401981 . 8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
00401984 . 56             PUSH ESI
00401985 . 8BF1           MOV ESI,ECX
00401987 . C646 0C 00     MOV BYTE PTR DS:[ESI+C],0
0040198B . 85C0           TEST EAX,EAX
0040198D .v75 63          JNZ SHORT 1.004019F2
0040198F . E8 02100000    CALL 1.00402996
00401994 . 8946 08        MOV DWORD PTR DS:[ESI+8],EAX
DS:[00415DF9]=7C8106C7 (kernel32.CreateThread)
```

```
EAX 7C960BE1 ntdll.RtlDecompressBuffer
ECX 7C917DE9 ntdll.7C917DE9
EDX 7C97B178 ntdll.7C97B178
EBX 7FFDF000
ESP 0012FE98
EBP 0012FF30
ESI 00000000
EDI 7C910208 ntdll.7C910208

EIP 00401962 1.00401962

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_PROC_NOT_FOUND (0000007F)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty -UNORM BDEC 01050104 006C006C
ST1 empty +UNORM 0069 002E0067 00620064
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                         ASCII
00415DAC  48 78 2A 6B 54 0D 67 70 79 69 7D 55 47 4F 56 7B  Hx*kT.gpyi}UGOV{
00415DBC  5F 6A 20 3A 36 45 57 25 34 59 46 29 51 26 6F 32  _j :6EW%4YF)Q&o2
00415DCC  2E 5A 39 53 3F 40 4E 6E 72 73 65 30 24 66 49 68  .Z9S?@Nnrse0$fIh
00415DDC  38 75 6D 43 50 2F 61 23 28 21 0A 63 2D 77 52 E1  8umCP/a#(!.c-wRß
00415DEC  0B 96 7C 7A 00 00 80 7C 4C E1 9A 80 7C C7 06 81  ♂û|z..Ç|LßÜÇ|╟♠ü
00415DFC  7C 94 2C 90 7C 22 4D 5C 6C 44 42 64 30 25 80 7C  |ö,É|"M\lDBd0%Ç|
00415E0C  4B 76 2C 3B 37 41 58 33 31 74 2B 00 00 00 00 00  Kv,;7AX31t+.....
00415E1C  00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ............☺...
00415E2C  01 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00  ☺....... .......
00415E3C  00 00 00 00 B0 20 33 00 00 00 00 00 00 00 00 00  ....░ 3.........
00415E4C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415E5C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415E6C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415E7C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415E8C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415E9C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415EAC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415EBC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00415ECC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0012FE98  00000000  pSecurity = NULL
0012FE9C  00000000  StackSize = 0
0012FEA0  004010FC  ThreadFunction = 1.004010FC
0012FEA4  7C800000  pThreadParm = kernel32.7C800000
0012FEA8  00000000  CreationFlags = 0
0012FEAC  00000000  pThreadId = NULL
0012FEB0  74696157
0012FEB4  53726F46
0012FEB8  6C676E69
0012FEBC  6A624F65
0012FEC0  00746365
0012FEC4  61657243
0012FEC8  68546574
0012FECC  64616572
0012FED0  00402800  1.00402800
0012FED4  7C800000  kernel32.7C800000
0012FED8  0012FEFC
0012FEDC  446C7452
0012FEE0  6D6F6365
0012FEE4  73657270
```

On December 13th 2019, @nao_sec discovered a new Exploit kit targeting users in Japan and it was given the name, Bottle Exploit Kit.

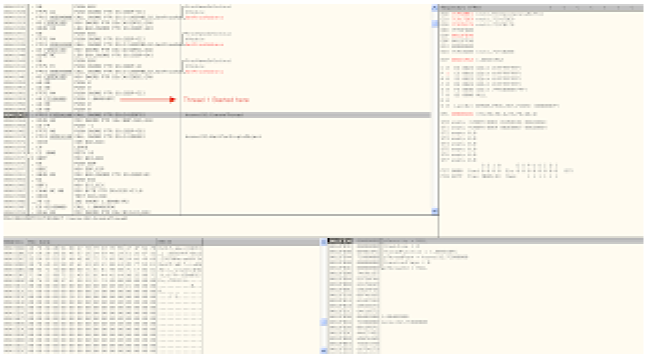
@nao_sec described in their blog the details of the Exploit Kit including the two vulnerabilities (CVE-2018-8174 and CVE-2018-15982) which were exploited in this attack.

In this article, I will go into the details of the multiple stages of unpacking the payload used in Bottle Exploit Kit.

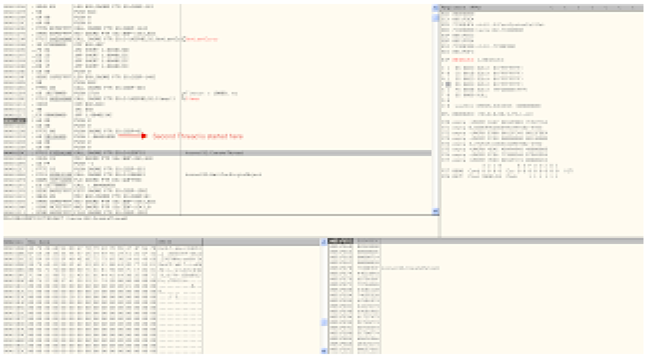
tl;dr: Multiple stages of packing are used in the payload. The XOR decryption routine used is common for all the payloads related to Bottle Exploit Kit and can be used to discover more instances.

MD5 hash of the sample discussed: ee98ef74c496447847f1876741596271

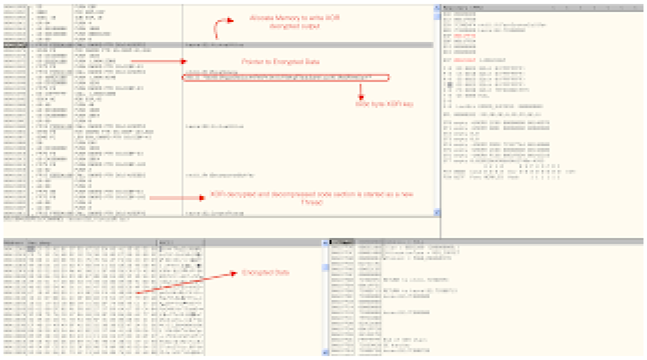
The WinMain() subroutine creates a new Thread as shown below:



The newly created Thread creates another Thread in turn as shown below:



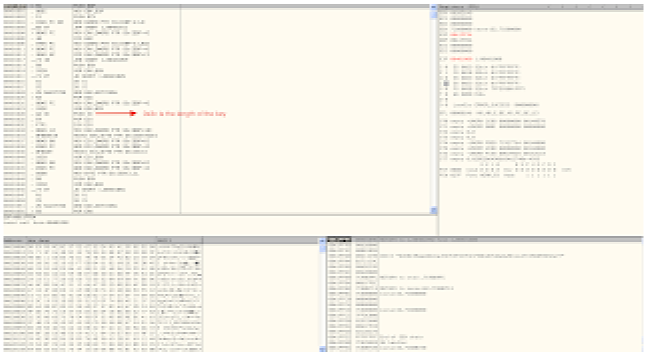
The first stage of unpacking is performed by Thread 2 (function start address: 0x40105b). The figure highlights the important stages of unpacking:


Allocate Memory to write XOR decrypted output
Pointer to Encrypted Data
0x3c byte XOR key
XOR decrypted and decompressed code section is started as a new Thread
Encrypted Data

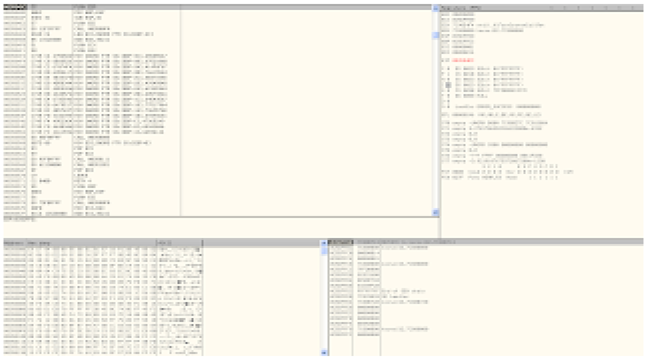
The main stages of unpacking of the first stage are:

1. VirtualAlloc() to allocate memory to decrypt stage 1.
2. RtlMoveMemory() to copy 0x1ed8 bytes of encrypted data from 0x412db8 to memory allocated in step 1.
3. XOR decryption routine at address: 0x401000 is invoked. The XOR decryption key is 0x3c bytes in length and is passed as an argument to the XOR decryption routine.
4. VirtualAlloc() is called again to decompress the XOR decrypted output of step 3.
5. Decompression is performed using RtlDecompressBuffer()
6. A new Thread with function start address set to decompressed code in step 5 is started.

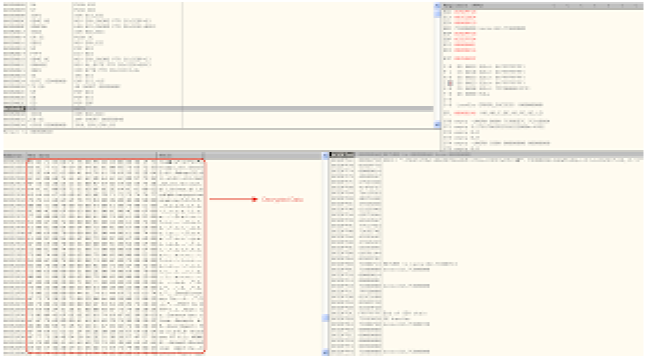
The XOR decryption routine mentioned in step 3 above is as shown below:



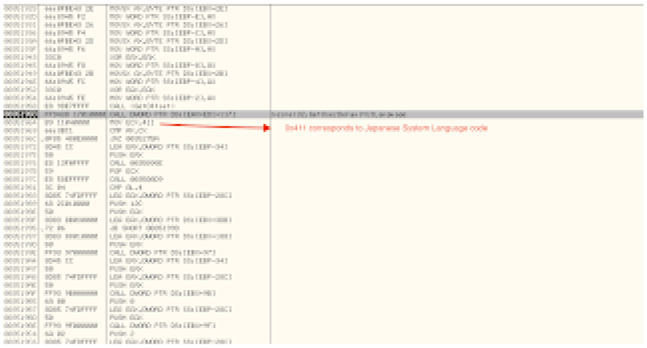
The next decrypted stage looks as shown below:



Another layer of XOR decryption is done by stage 2 which gives us the following decrypted data:



In the next stage of execution, it performs the System Language Check using the API, GetUserDefaultUILanguage() as shown below:



The system language code is compared with 0x411 which corresponds to Japanese System Language. The payload will execute completely only if the system language code is: 0x411.

Here is the list of decrypted strings:

```
0000000028DF   0000000028DF      0   shlwapi.dll
0000000028EB   0000000028EB      0   User32.dll
0000000028F6   0000000028F6      0   Advapi32.dll
000000002903   000000002903      0   ntdll.dll
00000000290D   00000000290D      0   Ws2_32.dll
000000002918   000000002918      0   Wininet.dll
000000002924   000000002924      0   Urlmon.dll
00000000292F   00000000292F      0   bcdfghklmnpqrstvwxzaeiouyT
000000002A56   000000002A56      0   DataDirectory %s
000000002A77   000000002A77      0   POST %s HTTP/1.1
000000002A89   000000002A89      0   Host: %s
000000002A93   000000002A93      0   Connection: close
000000002AA6   000000002AA6      0   Accept: */*
000000002AB3   000000002AB3      0   User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
000000002AE4   000000002AE4      0   Accept-Encoding: identity
000000002AFF   000000002AFF      0   Content-Type: application/x-www-form-urlencoded
000000002B30   000000002B30      0   Content-Length: %d
000000002B4C   000000002B4C      0   WSAStartup
000000002B57   000000002B57      0   socket
000000002B5E   000000002B5E      0   setsockopt
000000002B69   000000002B69      0   connect
000000002B7B   000000002B7B      0   closesocket
000000002952   000000002952      0   AppData\LocalLow
000000002974   000000002974      0   \Data\Tor
000000002988   000000002988      0   \Data\Tor\geoip
0000000029A8   0000000029A8      0   \Data\Tor\geoip6
0000000029CA   0000000029CA      0   \Tor\taskhost.exe
0000000029EE   0000000029EE      0   \Tor\tor.exe
000000002A08   000000002A08      0   \torrc
000000002A16   000000002A16      0   1.zip
000000002A22   000000002A22      0   1.exe
000000002A2E   000000002A2E      0   -o -qq "%s" -d "%s"
000000002A66   000000002A66      0   s-f "%s"
000000002B86   000000002B86      0   t.jpg
000000002B9B   000000002B9B      0   ALLUSERSPROFILE
000000002BBB   000000002BBB      0   rundll32.exe
```

From the Decrypted Strings above, we can see that it will make a Network Request to download TOR browser.

The XOR decryption routine used in the payload is the same among all the samples (DLL and EXE files) related to Bottle Exploit Kit instances.

Here are more MD5 hashes of payloads used in Bottle Exploit Kit:

5c9522927945f7fde17724360e9fec64
d408c3f58c3407e5d37ec524db03deb9
d6ae17d1d8ba79de1f936092297e44f9
d1bdc7c37f66702bf72d41a9276777dc
894794945683db1e708d2e9304821b19
6c71ca4978095b24a10487a84215d5bd
ee98ef74c496447847f1876741596271
e65322b4add2e5183616ce283e99614f
c67c6f2f212ffe7d99c7238c959c95e6
972d77bd40b0acfa9c0ffaf12b7cbba4
8fdd5e90c33b4feb83b74b3922c09c6d
e753d7a35a9144dd820d4d6e9be970ee

The only change is the XOR decryption key.

c0d3inj3cT