Flare On 7 - garbage

This write-up concerns the second challenge on Flare On 7 as I found it very nice to learn about PE files structure and basic reverse engineering.

Figure 1: Dashboard of Flare On 7.

The challenge starts with the following message : “One of our team members developed a Flare-On challenge but accidentally deleted it. We recovered it using extreme digital forensic techniques but it seems to be corrupted. We would fix it but we are too busy solving today's most important information security threats affecting our global economy. You should be able to get it working again, reverse engineer it, and acquire the flag.”. So it seems that the file is corrupted. Let’s put it in PE studio to see the properties of the file.

property	value
md5	CB85617125124F3FC945C7F375349DE3
sha1	FDD445057A5CE73444FC5C5AC50AC10AB0B44466
sha256	E30ED00A2763403BC0040F3EB5F6B22874892D9A79BCE5F4239404D6B9009B42
md5-without-overlay	n/a
sha1-without-overlay	n/a
sha256-without-overlay	n/a
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
first-bytes-text	M Z .. .. @ .. ..
file-size	40740 (bytes)
size-without-overlay	n/a
entropy	7.819
imphash	n/a
signature	UPX -> www.upx.sourceforge.net
entry-point	60 BE 00 F0 40 00 8D BE 00 20 FF FF 57 83 CD FF EB 10 90 90 90 90 90 90 8A 06 46 88 07 47 01 DB 75
file-version	n/a
description	n/a
file-type	**executable**
cpu	**32-bit**
subsystem	console
compiler-stamp	0x5EA2E073 (Fri Apr 24 05:49:55 2020)
debugger-stamp	n/a
resources-stamp	empty
exports-stamp	n/a
version-stamp	n/a

Figure 2: garbage.exe in PE studio.

As we can notice in Figure 2, the entropy and the signature of the file indicates that the file seems to be packed. Indeed, the entropy is quite high and the signature indicates that the file is UPX packed. Now, if we try to unpack the sample an exception occurs because the file is corrupted.

```
PS C:\Users\Malware_lab\Desktop\garbage_chal> upx.exe -d .\garbage.exe -o unpacked_garbage.bin
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96w       Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx: .\garbage.exe: OverlayException: invalid overlay size; file is possibly corrupt

Unpacked 1 file: 0 ok, 1 error.
```

Figure 3: Unpacking try.

Let's do some recon so we can identify how the file was corrupted. If we open the corrupted file with PE bear, we can notice that only the resource section seems to be corrupted. Opening the file with a hex editor, we see that a part of the manifest and the import address table are missing.

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of
UPX0	400	0	1000	E000	E0000080	0	0
UPX1	400	9A00	F000	A000	E0000040	0	0
.rsrc	9E00	124	19000	1000	C0000040	0	0

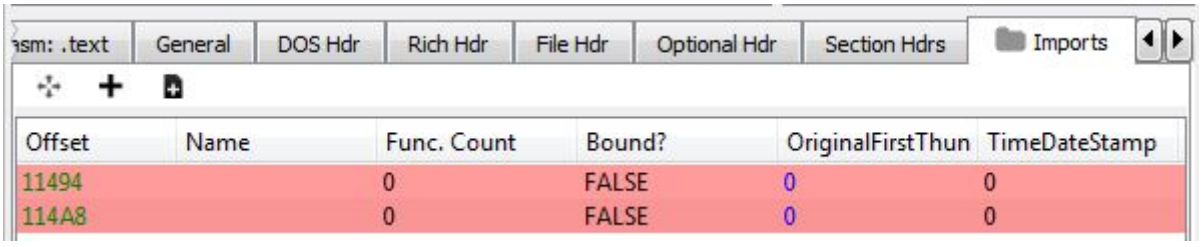
Figure 4: Section headers of garbage.exe.

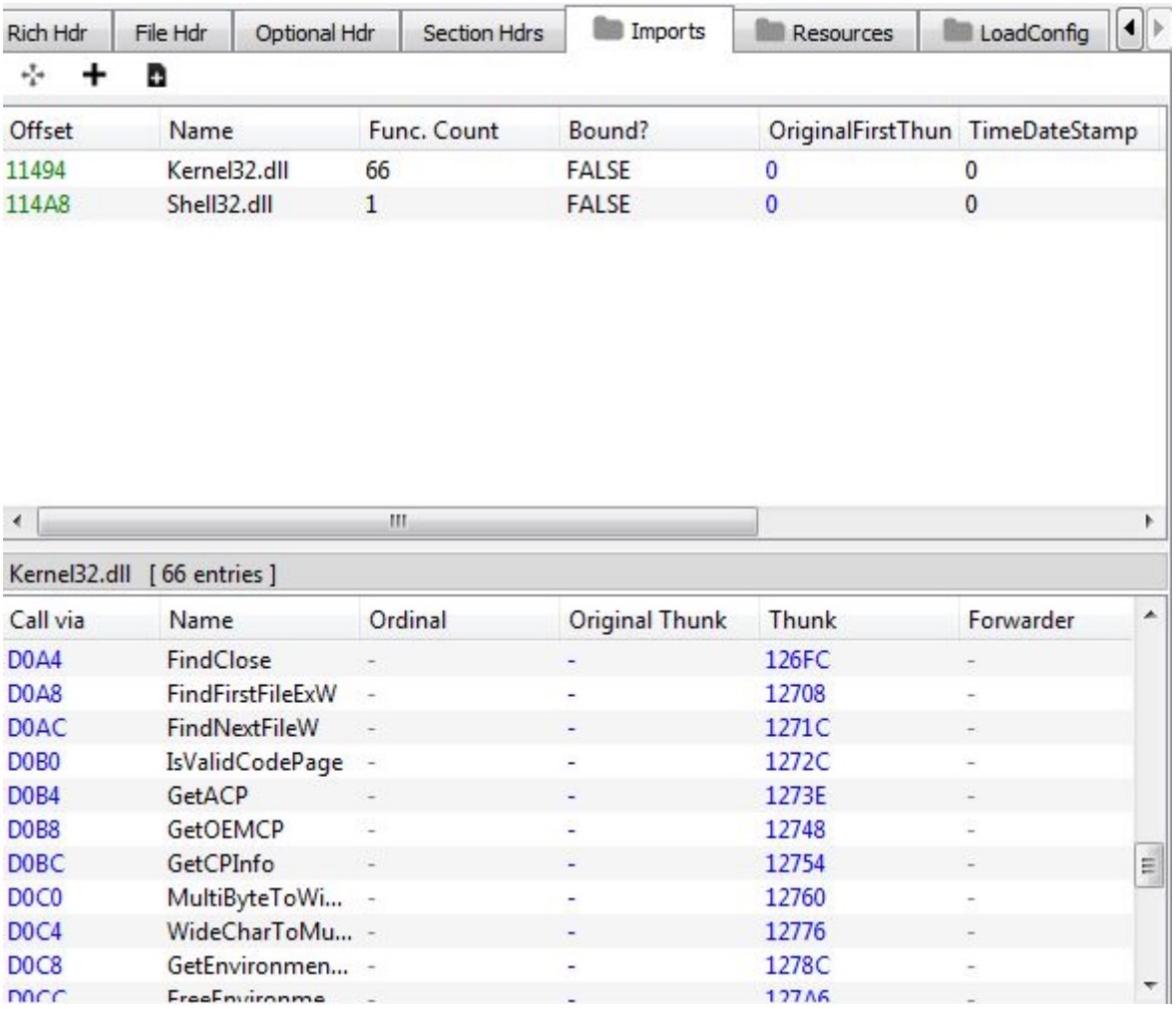
```
00009E30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00  ................
00009E40  09 04 00 00 48 00 00 00 5C 90 01 00 7D 01 00 00  ....H...\...}...
00009E50  00 00 00 00 00 00 00 00 60 50 01 00 3C 3F 78 6D  ........`P..<?xm
00009E60  6C 20 76 65 72 73 69 6F 6E 3D 27 31 2E 30 27 20  l version='1.0'
00009E70  65 6E 63 6F 64 69 6E 67 3D 27 55 54 46 2D 38 27  encoding='UTF-8'
00009E80  20 73 74 61 6E 64 61 6C 6F 6E 65 3D 27 79 65 73   standalone='yes
00009E90  27 3F 3E 0D 0A 3C 61 73 73 65 6D 62 6C 79 20 78  '?>..<assembly x
00009EA0  6D 6C 6E 73 3D 27 75 72 6E 3A 73 63 68 65 6D 61  mlns='urn:schema
00009EB0  73 2D 6D 69 63 72 6F 73 6F 66 74 2D 63 6F 6D 3A  s-microsoft-com:
00009EC0  61 73 6D 2E 76 31 27 20 6D 61 6E 69 66 65 73 74  asm.v1' manifest
00009ED0  56 65 72 73 69 6F 6E 3D 27 31 2E 30 27 3E 0D 0A  Version='1.0'>..
00009EE0  20 20 3C 74 72 75 73 74 49 6E 66 6F 20 78 6D 6C    <trustInfo xml
00009EF0  6E 73 3D 22 75 72 6E 3A 73 63 68 65 6D 61 73 2D  ns="urn:schemas-
00009F00  6D 69 63 72 6F 73 6F 66 74 2D 63 6F 6D 3A 61 73  microsoft-com:as
00009F10  6D 2E 76 33 22 3E 0D 0A 20 20 20 20 3C 73 65 63  m.v3">..    <sec
00009F20  75 72 69 74                                      urit
```

Figure 5: Hex view of the resource section.

After the rebuilding of the manifest and the imports in the garbage.exe file, we can finally unpack the file using the “upx -d “ command.

Figure 6: Manifest rebuilded





After code analysis, we can notice three important API calls in the main function.

```
*(undefined2 *)puVar3 = *(undefined2 *)puVar2;
FUN_00401000(local_13c,(int)&local_1c,0x14,(int)local_c4);
hFile = CreateFileA(local_13c[0],0x40000000,2,(LPSECURITY_ATTRIBUTES)0x0,2,0x80,(HANDLE)0x0);
FUN_00401045((int *)local_13c);
if (hFile != (HANDLE)0xffffffff) {
  local_140 = 0;
  FUN_00401000(local_13c,(int)&local_5c,0x3d,(int)local_12c);
  WriteFile(hFile,local_13c[0],0x3d,&local_140,(LPOVERLAPPED)0x0);
  FUN_00401045((int *)local_13c);
  CloseHandle(hFile);
  FUN_00401000(local_13c,(int)&local_1c,0x14,(int)local_c4);
  ShellExecuteA((HWND)0x0,(LPCSTR)0x0,local_13c[0],(LPCSTR)0x0,(LPCSTR)0x0,0);
  FUN_00401045((int *)local_13c);
}
uExitCode = 0xffffffff;
hFile = GetCurrentProcess();
TerminateProcess(hFile,uExitCode);
FUN_0040121b();
return;
```

Figure 6 : CreateFile, WriteFile, ShellExecuteA calls

Running the file in x32dbg and putting breakpoints on the three API calls, we notice the creation of the sink_the_tanker.vbs file and the writing of the MSGBOX VBScript command with our flag in it.

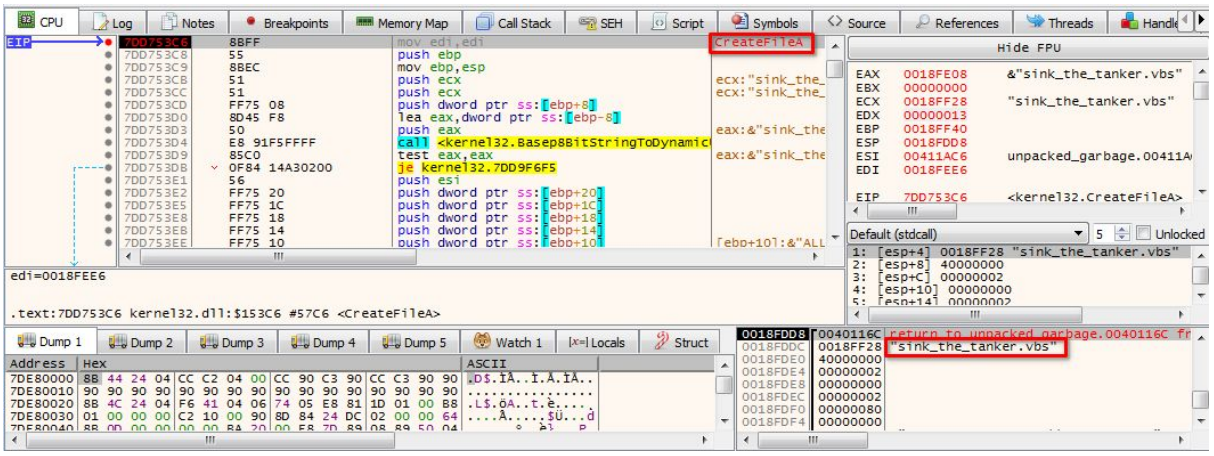


Figure 7 : CreateFileA call.

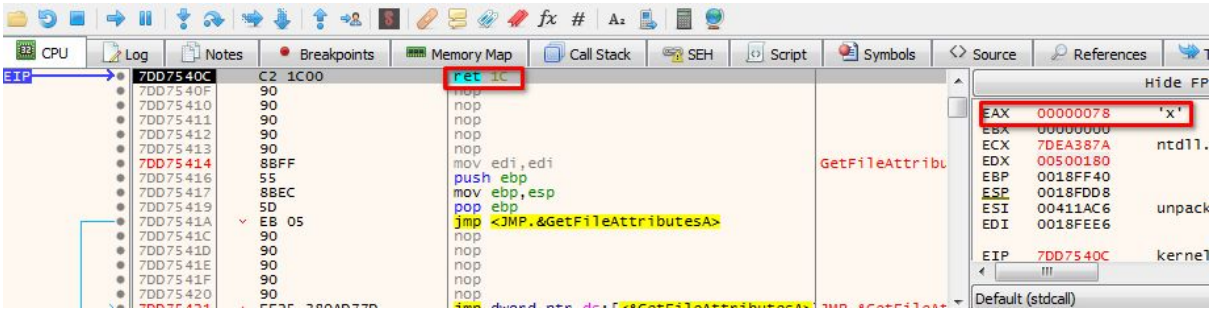


Figure 8 : Handle of sink_the_tanker.vbs file.

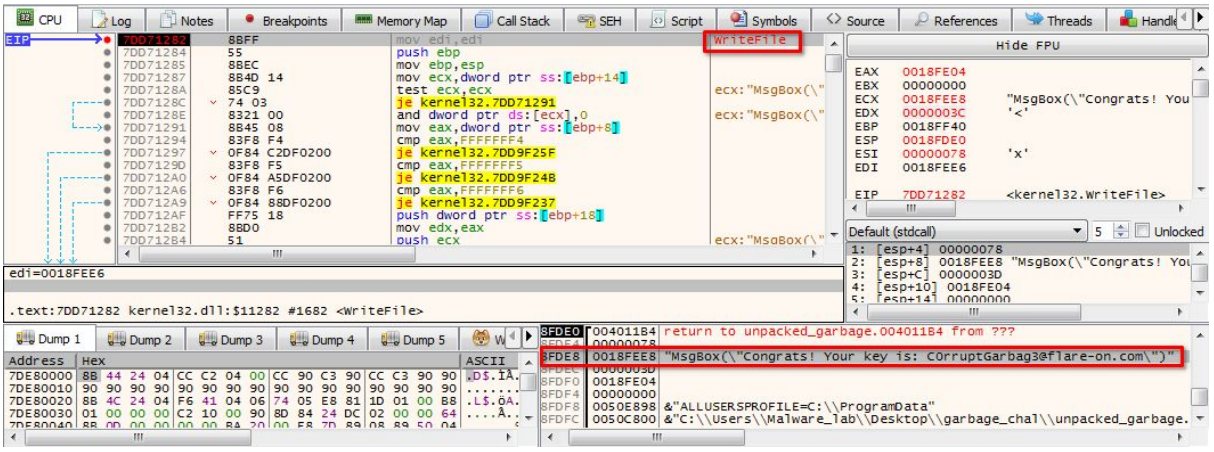


Figure 9 : WriteFile call.

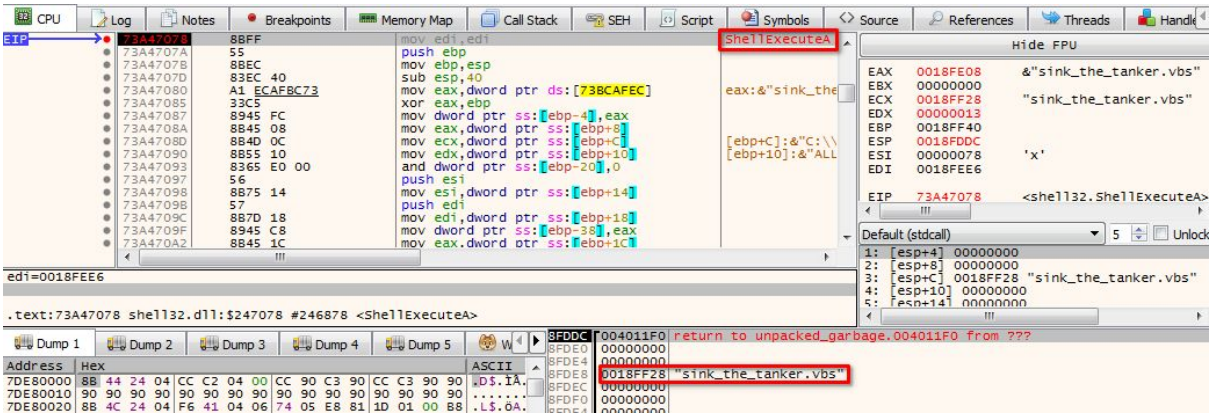


Figure 9 : ShellExecuteA call.