FIREEYE

FLARE

Flare-On 7: Challenge 2 –garbage.exe

Challenge Author: Jon Erickson

This challenge was inspired by a real-life experience of receiving and performing analysis of corrupt packed executable.

Performing on a quick triage on this sample reveals that it is packed with UPX. Performing strings analysis reveals some partial Base64 strings but not much else. Trying to unpack the file using the standard upx –d command reveals an error message as show in Figure 1.

\$ upx -d garbag	e-v1-prod.exe					
	Ultimate Pa	acker for eXe	cutables			
	Copyrigh	nt (C) 1996 -	2018			
UPX 3.95w	Markus Oberhumer,	Laszlo Molna	r & John Rei	ser Aug	g 26th 2018	
File si	ze Ratio	Format	Name			
upx: garbage-v1	-prod.exe: Overlay	Exception: in	valid overla	y size; f	file is possi	bly corrupt
Unpacked 1 file	: 0 ok, 1 error.					

Figure 1 - upx -d unpacking attempt

Since we cannot unpack this file statically, we can attempt to unpack it dynamically. Attempts to open run the binary in both x32dbg and WinDbg fail with indications that the file is not a valid application as shown in Figure 2.



Figure 2 - invalid application error message



Looking back more closely at the strings we encountered, we noticed the following XML data at the end of the file. This data should look familiar, many applications contain an XML manifest at the end. However, this manifest is not complete, it is truncated.



The truncated manifest is a clue that something is wrong with this file. Opening the file in a hex editor reveals that this partial XML data is at the end of the file, which means that the file itself it truncated. We can use the tool CFF explorer to examine the binary to determine what the PE file should be.

🛥 CFF Explorer VIII - [garbage-v1-prod.e	xe]						
File Settings ?							
	garbage-v1-p	rod.exe	×				
	Property	Value					
File: garbage-v1-prod.exe	File Name	C:\Users	\user\Desktop\garbage-v1-prod.exe				
	File Type	Portable Executable 32					
File Header	File Info	UPX 2.90	[LZMA] (Delphi stub) -> Markus Oberhumer, Laszlo Molnar & John Reise				
Data Directories [x]	File Size	39.79 KB	(40740 bytes)				
Section Headers [x] Besource Directory	PE Size	40.50 KB	(41472 bytes)				
- Address Converter	Created	Tuesday	02 June 2020, 08.15.59				
Dependency Walker Mex Editor	Modified	Friday 24 April 2020, 07.59.45					
- Martifier	Accessed	ssed Tuesday 02 June 2020, 08.15.59					
	MD5 CB85617125124F3FC945C7F375349DE3						
	SHA-1 FDD445057A5CE73444FC5C5AC50AC10AB0B44466						
UPX Utility	Property	Va	lue				
	Empty	No	additional info available				
L	,						

Figure 3 - CFF Explorer checking for truncation



CFF Explorer reveals that the PE size is 41,472 bytes, however the file size is only 40,740 as shown in Figure 3. Meaning the file is truncated by 732 bytes. While this is a small number of bytes missing it causes the Windows loader to fail and the UPX utility to error.

There are many solutions to this challenge. I am going to briefly walk through two solutions.

Unpacking solution #1

The first solution was pointed out to me by one of my FLARE colleagues, and is a very simple. Since the PE file has been truncated and is missing 732 bytes, we can append 732 NULL bytes to the end of the file. We can now use the standard UPX utility to unpack the padded binary. While this new unpacked file does not run, it will still allow us to perform static analysis.

Running strings on the unpacked binary does not reveal much, just two large base64 strings. Now that we have the unpacked file, we can try to determine its purpose. This will be discussed later in this paper.

Unpacking solution #2

The second solution is more compliant than the first. I am providing the details so that others can learn from it. When taking a closer look at the Data Directories of the original file in CFF Explorer we see two areas that stand out in red as shown in Figure 4.

✓ CFF Explorer VIII - [garbage-v1-prod.e:	xe]										
File Settings ?											
💫 🤳 👘	garbage-v1-prod.exe										
7	Member	Offset	Size	Value	Section						
File: garbage-v1-prod.exe Sos Header	Export Directory RVA	00000170	Dword	0000000							
- 🖓 🗉 Nt Headers	Export Directory Size	00000174	Dword	00000000							
File Header Optional Header	Import Directory RVA	00000178	Dword	000191DC	Invalid						
Data Directories [x]	Import Directory Size	0000017C	Dword	000000C0							
Esction Headers [x] Escurce Directory	Resource Directory RVA	00000180	Dword	00019000	UPX0						
Address Converter	Resource Directory Size	00000184	Dword	000001DC							
	Exception Directory RVA	00000188	Dword	0000000							
- Silvertifier	Exception Directory Size	0000018C	Dword	0000000							
	Security Directory RVA	00000190	Dword	0000000							
	Security Directory Size	00000194	Dword	0000000							
	Relocation Directory RVA	00000198	Dword	0001929C	Invalid						
	Relocation Directory Size	0000019C	Dword	0000010							
	Debug Directory RVA	000001A0	Dword	0000000							
	Debug Directory Size	000001A4	Dword	0000000							
	Architecture Directory RVA	000001A8	Dword	00000000							

Figure 4 - CFF Explorer invalid directories

FireEye, Inc. | 601 McCarthy Blvd. Milpitas, CA 95035 | 408.321.6300 | 877.FIREEYE (347.3393) info@fireeye.com | www.FireEye.com



The reason these directories are marked in red is because the data for these directories exists within the truncated area of the file. This truncation occurs within the .rsrc section of the file. To get this file to run, we can remove the .rsrs section from the file using CFF Explorer. After doing so we also need to zero out the Import Directory, Resource Directory, and Relocation Directory from the Data Directories. Once all changes have been made, we can save our changes as a new filename.

Double clicking the resulting file results in a crash. However, it runs. At this point we can attempt to load the new file into x32dbg and examine the crash.

🕷 garbage-v1-p	rod-no-rescs.exe -	PID: 2EC - Module: garbage	-v1-prod-no-rescs.exe - Thread: Main Thread 1F4 - x	2dbg [Elevated]		
<u>File View D</u> eb	ug <u>T</u> race <u>P</u> lugin	ns Favourites <u>O</u> ptions <u>F</u>	<u>l</u> elp Apr 29 2019			
📄 🔊 🔳 🄿	11 🕈 み 🖄	👌 🎍 💲 🛥 📓 🥖	😓 🛷 🥒 fx # Az 🖺 🗐 👳			
🕮 СРИ 🏾 🌳	Graph 🛛 🛃 Log	🖺 Notes 🔹 Breakpo	ints 🛛 🛲 Memory Map 📄 Call Stack 🛛 📆 SEH	💿 Script 🛛 🗎 Syr	mbols <> Source	🖓 References 🛛 🐭 Threads 🕴 🕑
>	00418866 0041886C	8DBE 00600100 8B07	<pre>lea edi,dword ptr ds:[esi+16000] mov eax,dword ptr ds:[edi]</pre>	^	Hide FPU	
	0041886E 00418870 ~ 00418872	09C0 74 3C 8B5F 04 8D8430 DC810100	or eax,eax je garbage-v1-prod-no-rescs.4188AE mov ebx,dword ptr ds:[edi+4] lea eax dword ptr ds:[eax+esi+18100]	edi+4:"CurrentF	EAX 00419234 EBX 0040D000 ECX 00000000	garbage-v1-prod-no-rescs
	0041887C 0041887E 0041887F	01F3 50 83C7 08	add ebx,esi push eax add edi,8		EDX 00417D57 EBP FFFFFF9 ESP 0018FF68	garbage-v1-prod-no-rescs
	00418882 00418888 00418889 00418889 00418888 00418886	FF96 18820100 95 8A07 47 08C0	<pre>Call dword ptr ds:[est+18218] xchg ebp,eax mov al,byte ptr ds:[edi] inc edi or al,al</pre>		EIP 00418882	garbage-v1-prod-no-rescs garbage-v1-prod-no-rescs garbage-v1-prod-no-rescs
	00418890 00418892 00418893 00418894 00418894	89F9 57 48 F2:AE	pe garbage v profilo resestatobe movieck, edi push edi dec eax repne scasb		EFLAGS 00010202 ZF 0 PF 0 AF 0 OF 0 SF 0 DF 0 CF 0 TF 0 IF 1	
	00418897 00418890 00418890 0041889F 004188A1 004188A3	FF96 20820100 09C0 74 07 8903 83C3 04	<pre>cal dword ptr ds:[esi+18220] or eax,eax ie garbage=v1-prod-no-rescs.4188A8 mov dword ptr ds:[ebx],eax add ebx,4</pre>		LastError 000000 LastStatus 000000 GS 002B FS 0053 ES 002B DS 002B	00 (ERROR_SUCCESS) 00 (STATUS_SUCCESS)
	004188A6 004188A8 (Default (stdcall)	▼ 5 🚔 🗖 Unlocked			
dword ptr [es	i+18218]=[4192 garbage-v1-pr	2: [esp+4] 000000 3: [esp+8] 000000 4: [esp+C] 0018FF 5: [esp+10] 0018FF	00 00 94 58C			

Figure 5 - x86dbg crash analysis

We can see in Figure 5 that the crash occurs when trying to call a function. Knowing that this sample was packed with UPX we can examine the source code of the UPX unpacking stub to see where this crash has occurred. (https://github.com/upx/upx/blob/d7ba31cab8ce8d95d2c10e88d2ec787ac52005ef/src/stub/src/i386-

<u>win32.pe.S#L99</u>). Looking closely at the source reveals that the program is attempting to call LoadLibrayA. It is unfortunate that the sample crashes, but what do you expect when you remove a whole section? The great thing is that it crashes in the unpacking stub after decompression. It crashes during import resolution. Therefore, we can perform a memory dump and perform analysis on the unpacked code.

Sample analysis, the easy way

Now that we have an unpacked sample, we can start to solve this challenge. For this section we will be using the unpacked code extracted from section #1 above. The first thing to try is executing the unpacked sample. We encounter an error message te4lling us the side-by-side configuration is incorrect as shown in Figure 6 below.



C:\User:	s\user\Desktop\garbage-v1-prod-with-padding.exe	8
\otimes	C:\Users\user\Desktop\garbage-v1-prod-with-padding.exe	
	The application has failed to start because its side-by-side configuration is incorrect. Pl see the application event log or use the command-line sxstrace.exe tool for more detai	lease I.
		ОК

Figure 6 - Bad side-by-side configuration

This is an artifact of the truncation. Part of the side-by-side configuration is missing. To verify this, we can load the unpacked sample into CFF explorer and view its resources. Figure 7 below shows the truncated configuration. We can remove this resource by right clicking and selecting the remove resource option.

garbage-v1-prod-with-pa	dding.
Configuration Files	<pre><?xml version = '1.0' encoding = 'UTF-8' standalone = 'yes'?> <assembly manifestversion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1"> <trustinfo xmlns="urn:schemas-microsoft-com:asm.v3"> <securit< pre=""></securit<></trustinfo></assembly></pre>

Figure 7 - Truncated resource

Saving the executable and attempting to run the sample again reveals another error message shown in Figure 8 below.



Figure 8 - Missing .DLL



Using CFF explorer again we can examine the import table. As you can see in Figure 9 below, the module names are missing. This is an artifact of the file being corrupt, even thought the upx –d command worked. We can infer the module names by looking at the functions to resolve in the lower portion of the imports view.

garbage-v1-p	rod-wi	th-padding-r	10-res	ource.exe						
Module Name		Imports		OFTs		TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)	
00011634	0011634 N/A 00011494			00011498	0001149C	000114A0	000114A4			
szAnsi		(nFunction	ns)	Dword		Dword	Dword	Dword	Dword	
		66		00000000		0000000	0000000	00012434	0000D000	
		1		00000000		0000000	0000000	00012452	0000D10C	
OFTs	FTs (IAT)	Hint		Nam	ne				
Dword	Dwo	rd	Wor	d	szAnsi					
N/A	0001	23E4	0000		GetC	CurrentProcess				
N/A	0001	23F8	0000		Write	eFile				
N/A	0001	2404	0000		Term	ninateProcess				

Figure 9 - Missing module names

You can use CFF explorer to populate the module name fields as shown in Figure 10 below.

garbage-v1-	prod-wi	ith-padding-	no-res	ource.exe					
Module Name		Imports		OFTs		TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
00011652		N/A		000114A8		000114AC	000114B0	000114B4	000114B8
szAnsi		(nFunction	ns)	Dword		Dword	Dword	Dword	Dword
kernel32.dll		66		00000000		0000000	0000000	00012434	0000D000
shell32.dll	shell32.dll 1		00000000		0000000	0000000	00012452	0000D10C	
OFTs	FTs	(IAT)	Hint		Nam	ie			
Dword	Dwo	ord	Wor	d	szAn	isi			
N/A	0001	2442	0000		ShellExecuteA		_		

Figure 10 - Fixed module names



After fixing the module names in CFF explorer and saving, we now have a working executable, which gives us the flag as shown below in Figure 11.

×
Congrats! Your key is: C0rruptGarbag3@flare-on.com
ОК

Figure 1	11 - 1	The F	lag
----------	--------	-------	-----

Sample analysis, the hard way

If we attempt to open the unpacked sample in IDA we quickly realize that something isn't quite right. The imports are all missing as shown below in Figure 12. The red text indicates where there would be a call to an imported function however IDA does not know which ones.



```
v12[0] = 741548835;
v12[1] = 1231306510;
strcpy(v10, "nPTnaGLkIqdcQwvieFQKGcTGOTbfMjDNmvibfBDdFBhoPaBbtfQuuGWYomtqTFqvBSKdUMmciqKSGZaosWCSoZlcIlyQpOwkcAgw ");
v12[2] = 67771914;
v12[3] = 436344355;
v12[4] = 604530244;
strcpy(v11, "Kg1PFOsQDxBPXmc10pmsdLDEPMRWbMDzwhDGOyqAkVMRvnBeIkpZIhFznwVylfjrkqprBPAdPuaiVoVugQAlyOQQtxBNsTdPZgDH ");
v12[5] = 745804082;
v12[6] = 255995178;
v12[7] = 224677950;
v12[8] = 387646557;
v12[9] = 84096534;
v12[10] = 134815796;
v12[11] = 237248867;
v12[12] = 1479808021;
v12[13] = 981018906;
v12[14] = 1482031104;
v13 = 84;
v14[0] = 989990456;
v14[1] = 874199833;
v14[2] = 1042484251;
v14[3] = 1108412467;
v14[4] = 1931350585;
sub 401000(v14, 20, v11, 0);
v3 = MEMORY[0x12418](v9[0], 0x40000000, 2, 0, 2, 128, 0);
sub_401045(v9);
if ( v3 != -1 )
{
  v8 = 0;
  sub_401000(v12, 61, v10, v4);
  MEMORY[0x123F8](v3, v9[0], 61, &v8, 0);
  sub 401045(v9);
       RY[0x12426](v3);
  sub_401000(v14, 20, v11, v5);
MEMORY[0x12442](0, 0, v9[0], 0, 0, 0);
  sub 401045(v9);
}
 6 = \frac{MEMORY[0 \times 123E4]}{(-1)};
        ev12404](v6);
```

Figure 12 - Hexrays with missing imports

At this point we could try to infer the imports based on their usage, but there is a better way. We can use the same technique as discussed in the previous section using CFF explorer to add the missing module names.

Viewing the code now in Hexrays shows a very simple program which opens and writes to a file, which it then executes using the ShellExecute API.

To figure out what filename and contents are used we first need to understand the decoding routine sub_401000. Looking at the function we can see that is performs a multi-byte xor of a buffer using a key (large Base64 strings).

Decoding the data which will be used for the CreateFileA API reveals the filename sink_the_tanker.vbs

Decoding the data which will be written to the vbs script reveals the string MsgBox("Congrats! Your key is: C0rruptGarbag3@flare-on.com")

This is the exact same string which was displayed in the last section after executing the sample!

Conclusion



This challenge illustrated a real issue which I have come across during my day job, analysis of a corrupt packed executable. As you can see it is still possible in certain situations to perform analysis of corrupt files. Thanks for playing!