# Manual Unpacking Of Upx Packed Executable Using Ollydbg and Importrec

## Asha Devi, Gaurav Aggarwal

Independent Security Researcher SecurityExploaded and M.Tech Student of ITM UNIVERSITY, Gurgaon (Haryana)INDIA 122001.

**Summary:** A 'Packer' is a compression routine that compress an executable file. Packers are used on executable for two main reasons: to shrink programs or to thwart detection or analysis. When malware has been packed, an analyst typically has access to only the packed file, and cannot examine the original unpacked program or the program that packed the malware. In order to unpack an executable, we must undo the work performed by the packer, which requires that we understand how a packer operates. All packers take an executable file as input and produce an executable file as output. The packed executable is compressed, encrypted, or otherwisetransformed, making it harder to recognize and reverse-engineer. Unpacked executable are loaded by the OS. With packed programs, the **unpacking** stub is loaded by the OS, and then the unpacking stub loads the original program. The code entry point for the executable points to the unpacking stub rather than the original code. The original program is generally stored in one or more extra sections of the file. **Key words:** Packing, Unpacking stub, PE header, Sections

### I. Introduction:

UPX unpacking is a process of decompressing the packed executable and reconstructs its import address table. When an Executable or DLL is packed its IAT table destruct and need to construct during unpacking process. Unpacked executable or DLL are loaded by the operating system. But when the program is packed, only the unpacking stub is loaded by the O.S and then unpacking stub load the original program. The code entry point points to the Unpacking stub then OEP.

The unpacking stub can be viewed by the malware analyst and understanding different parts of stub is necessary to unpack the executable. The unpacking stub is often small and its main functionality is to **unpack the Original Executable**.

The unpacking stub perform the following steps:

- Unpack the original executable into memory.
- Resolve all the imports of original executable.
- Transfer execution to the Original Entry point.



Fig 1: Original Executable prior to Packing



Fig 2: Packed Executable after original code is packed and unpacking stub added.

## II. Theoretical Consideration:

When the UPX packed exe will execute followings will happen:

- Execution starts from OEP.
- First it saves the current register status using PUSHAD instruction
- All the packed sections are unpacked into memory
- Resolve the import table of original executable file
- Restore the original register status using POPAD instruction
- Finally jump to the original entry point to begin actual execution.

The instruction that transfers execution to the OEP is commonly referred to as the *tail jump*. A jump instruction is the simplest and most popular way to transfer execution. Since it's so common, many malicious packers will attempt to changethis function by using a **ret**or **call**instruction. Sometimes the tail jump is obscured with OS functions that transfer control, such as NtContinueor ZwContinue.



Fig 3: The program after being unpacked and loaded into memory and there is no import.



Fig 4: The fully unpacked program. The import table is reconstructed and starting point is backto OEP.

#### **Identifying Packed Programs:**

There are few indicators which show that the file is packed or not:

- The program has few imports, and particularly if the only imports are LoadLibraryand GetProcAddress.
- When the program is opened in IDA Pro, only a small amount of code is recognized by the automatic analysis.
- When the program is opened in OllyDbg, there is a warning that the program may be packed. The program shows section names that indicate a particular packer (such as UPX0).
- The program has abnormal section sizes, such as a .text section with a Size of Raw Data of 0 and Virtual Size of nonzero.
- Packer-detection tools such as PEiD, ExeScan can also be used to determine if an executable is packed.

#### Entropy Calculation

Packed executables can also be detected via a technique known as *entropycalculation*. Entropy is a measure of the disorder in a system or program.Compressed or encrypted data more closely resembles random data, and therefore has high entropy; executables that are not encrypted or compressed have lower entropy.

#### **Unpacking Options:**

There are mainly three options for the unpacking of executable:

- Automated static unpacking,
- Automated dynamic unpacking
- Manual dynamic unpacking
- I will explain Manual Unpacking:

#### Manual Unpacking:

Sometimes, packed malware can be unpacked automatically by an existing program, but more often it must be unpacked manually. Manual unpacking can sometimes be done quickly, with minimal effort; other times it can be a long process.

There are two common approaches to manually unpacking a program:

- Discover the packing algorithm and write a program to run it in reverse. By running the algorithm in reverse, the program undoes each of the steps of the packing program. There are automated tools that do this, but this approach is still inefficient, since the program written to unpack the malware will be specific to the individual packing program used. So, even with automation, this process takes a significant amount of time to complete.
- Run the packed program so that the unpacking stub does the work for you, and then dump the process out of memory, and manually fix up the PE header so that the program is complete. *This is the more efficient approach.*

#### • *Rebuilding the Import Table with Import Reconstructor:*

Rebuilding the import table is complicated, and it doesn't always work in OllyDump. The unpacking stub must resolve the imports to allow the application to run, but it does not need to rebuild the original import table. When OllyDbg fails, it's useful to try to use Import Reconstructor (ImpRec) to perform these steps.

#### **III.** Experimental Consideration:

In order to unpack the packed program need the following steps:

1. Load the UPX packed EXE into PEid or ExeScan to confirm that it is UPX Packed.



Fig 5: PEid result showing UPX Packed File.

#### ExeScanConfirmed that it is packed.

*						
[+] File: C:\Documents and Settings\Administrator\Desktop\PackedProcmon.exe						
[*] MD5 : e37914a11c449b110b089b4a9cc260b8 [*] SHA-1 : c6482796af630d7ff419b03357a305286a2989f5 [*] SHA-256 : 1b30c05d68e2290f4c4a75c8870c44933c65b1c47c157ad64aabe1 l66f3bcaa7						
[+] File Type: EXE 🥢						
[+] Signature [Commilter/Packer]						
['UPX 2.70 [LZMA] -> Markus Oberhumer, Laszlo Molnar & John Reiser']						
[+] Address of entry point : 0x0026a240						
[+] Image Base Address : 0x00400000						
I+] Sections Name: UPX0 Virtual Address: 0x00001000 Size: 0x001aa000 Entropy: 0.000000 Name: UPX1 Virtual Address: 0x001ab000 Size: 0x000c0000 Entropy: 7.937702 Name: .rsrc Virtual Address: 0x0026b000 Size: 0x00008000 Entropy: 5.216060						
[+] Anomalies Check						
<pre>[*] Based on the sections entropy check! file is possibly packed [*] Header Checksum is zero! [*] Enrty point is outside the 1st(.code) section! Binary is possibly pa cked</pre>						
[+] Following expected Malware APIs are Detected						
[-] Import Table						
IA: 0x00672a20 GetProcAddress IA: 0x00672a1c LoadLibraryA IA: 0x00672a28 VirtualAlloc IA: 0x00672a24 VirtualProtect						
[-] Entire Executable						
1 times GetProcAddress 1 times LoadLibrary 1 times LoadLibraryA 1 times VirtualAlloc						

Fig 6: ExeScan Result showing that PackedProcmon.exe is a packed file.

#### 2. Load the UPX Packed EXE into Ollydbg.

The first instruction is usually PUSHAD, before starting decompression routine, it saves all the register content. Now the instructions following are basically decompressing original code.



**3** : Now we reach PUSHAD,put a hardware breakpoint so as to stop at POPAD instruction. This will help us to stop execution when POPAD instruction is executed later on.

Regi	ster	S (FFU)	× ×	75 07			bco
		0000			Backup	•	269
		FFB0	1D-+	. 83EE F(	Сору	۲	
		1 <mark>E4F4</mark> ntdll.KiFastSystemCal 14000	INCT			۲	58
ESP	0011	CEC4		. B8 0100 > 01DB .∨75 07	Breakpoint	≯	Memory, on access
	00: FFF	Increment	Plus	. 881E . 83EE FO	Search for	Þ	Memory, on write
	703	Decrement	Minus	. 11DB	Follow DWORD in Disassembler		
EIP	006			> 11C0 . 01DB	Follow DWORD in Dump		Hardware, on access   Byte
сø		Zero		. ×73 0B . ×75 28	Go to	•	Hardware, on write   Word
P 1	ECS EF	Set to 1		. 8B1E		_	Hardware, on execution Dword
A Ø	S	Modify	Enter	. 83EE F0 . 11DB	✓ Hex	),	
Z 1 S 0	E8			.∨72 1F > 48	Text	۲	AB
S Ø T Ø	Gŝ	Copy selection to clipboard	Ctrl+C	01DB	Short	۲	298
DØ		Copy all registers to clipboard		. 881E . 83EE F0	Long	۲	n
0 0	_L4			. 11DB > 11C0	Float	۲	
EFL	000	Follow in Dump		> 1100	Disassemble		170
	emp	•			Special		
	emp	Follow in Stack			special	<i>.</i>	
012	ent					·	

#### 1. Now execute the program.Program execution stop at Breakpoint.

🗮 OllyDbg - PackedProcr	mon.exe - [CPU - main thread, module Packe
C File View Debug Plugin	ns Options Window Help
	H } ↓ → → L E M T W H C
005673E7         004424         80           006673E8         > 6A         00           006673E0         - 3954         .           006663EF         - 75         FA           006663E7         - 835C         80           006663F4         - E9         78B9E4FF           006663F4         - E9         78B9E4FF           006663F4         - 00         00           006663F4         - 00         00	LEA EAX,DWORD PTR SS:[ESP-80] PUSH 0 CMP ESP,EAX JNZ SHORT PackedPr.006663EB SUB ESP,-80 JMP PackedPr.00485D73 DB 00 DD PackedPr.00480000 DD PackedPr.00480000 DD PackedPr.00480000

2. Now Search for Jmp instruction which will take us to the actual OEP in the original program. At this point we know the decompression stub is executed and ready to jump at original code.

💥 OllyDbg - PackedProci	mon.exe - [CPU - main thread, module Packe
C File View Debug Plugir	ns Options Window Help
	₩ ¥ ↓ → → LEMTWHC
0066A3E7         .8D4424         80           0066A3EB         > 6A         00           0066A3EF         .75         FA           0066A3EF         .75         FA           0066A3E7         .83EC         80           0066A3E7         .924         0066A3E7           0066A3E7         .83EC         80           0066A3E7         .00         0066A3E7           0066A3E7         00         00004800           0066A3E7         .00         00004800	LEA EAX,DWORD PTR SS:[ESP-80] PUSH 0 CMP ESP,EAX JNZ SHORT PackedPr.0066A3EB SUB ESP,-80 JNP PackedPr.004B5D73 DB 00 DD PackedPr.00480000 DD PackedPr.00480000 DB 00

3. Now we reach at original OEP, now it's time to dump the process.



4. Now if we try to open the UnpackedProcmon.exe it gives error because IAT table is not constructed.



5. Now next step is to Reconstruct the IAT using importREC. Attach the Packedprocmon and get imports and Fix Dump to Unpackedprocmon.exe.





## 7. When we double click UnpackedProcmon.exe ,we have following result. IAT has been fixed.

File Edit Event Filter To	ols Options Help	
🖻 🖶 🕴 🍳 🅦 🖾	- 🗟 🗛 🔄	👭 📕   🌋 🔒
Time Process Name	PID Operation	Path
7:15:2 👌 UnpackedProc	2636 🌋 RegOpen Key	HKLM\System\CurrentConti
7:15:2 👌 UnpackedProc	2636 🌋 RegQueryValue	HKLM\System\CurrentConti
7:15:2 👌 UnpackedProc	2636 🌋 RegQueryValue	HKLM\System\CurrentConti
7:15:2 🚖 UnpackedProc	2636 🌋 RegQueryValue	HKLM\System\CurrentCont
7:15:2 👌 UnpackedProc	2636 🌋 RegQueryValue	HKLM\System\CurrentCont
7:15:2 🚖 UnpackedProc	2636 🌋 RegCloseKey	HKLM\System\CurrentCont
7:15:2 🚖 UnpackedProc	2636 🧟 Thread Create	
7:15:2 🚖 UnpackedProc	2636 🧟 Thread Create	
7:15:2 🚖 UnpackedProc	2636 🧟 Thread Create	
7:15:2 🚖 UnpackedProc	2636 🧟 Thread Create	
	2020 TI 10 1	

#### IV. Conclusion:

In an experiment observation we found that we can manually unpack UPX packed file and reconstruct IAT table using Ollydbg and importRec .

## Program result before unpacking:

6.

[+] File: C:\Documents and Settings\Administrator\Desktop\PackedProcmon.exe
[*] MD5 : e3791411c449b110b089b4a9cc260b8 [*] SHA-1 : c6482796af630d7ff419b03357a305286a2989f5 [*] SHA-256 : 1b30c05d68e2290f4c4a75c8870c44933c65b1c47c157ad64aab l66f3bcaa7
[+] File Type: EXE
[+] Signature [Compiler/Packer]
['UPX 2.50 [LZMA] -> Markus Oberhumer, Laszlo Molnar & John Reiser']
[+] Address of entry point : 0x0026a240
[+] Image Base Address : 9x90480000
[+] Sections Name: UPX0 Virtual Address: 0x00001000 Size: 0x001aa000 Entropy: 0.0000000
Entropy: 7.937702 Entropy: 7.937702
Entropy: Name:prc Uirtual Address: 0x0026b000 Size: 0x00008000 Entropy: 5.216060
[+] Anomalies Check
[*] Based on the sections entropy check! file is possibly packed [*] Headder Checksum is zero! [*] Enrty point is outside the 1st(.code) section! Binary is possibly ]
FED v0.95
ile: C\Documents and Setting\Administrator\Desktop\PackedProcmon.e
[-] Import Table
Intrypoint: 0026A240 EP Section: UPX1 > IA: 0x00672a20 GetProcAddress
lie Offset: 0000F640 First Bytes: 60,8E,00,8D ≥ 10.2 M06722a1c LocalLibyearv04 IA: 0×00672a28 VirtualAlloc
inkerInfo: 9.0 Subsystem: Win32 GUI ➤ IA: 9.001672.a2.4 UirtualProtect
[-] Entire Executable
UFX0.637.6-1.02/1.05-2.90-> Markus&Laszlo[Overlay] 1 times GetProcAddress
MultiScan Task Viewer Options About Exit 1 times Local Library
Stay on top >> 1 times Lighting Afgh

#### **Program Result after Unpacking:**

羅 PEiD v0.95						
File: C:\Documents and Settings\Admi	inistrator\Desktop\UnpackedProcmor					
Entrypoint: 000B5D73	EP Section: UPX0	>				
File Offset: 000B5D73	First Bytes: E8,B4,A3,00	>				
Linker Info: 9.0	Subsystem: Win32 GUI	>				
Nothing found * Multi Scan Task Viewer O ✓ Stay on top	ptions About E	×it				
	C:\WINDOWS\system32\cr	nd.exe				- 8
<pre>[*] File: C:\Documents and Settings\Administrator\Desktop\UnpackedProcmon.ex [*] MD5 : 05f9830fa5f95d7f82515982f53fcdbb [*] SHA-1 : e0a53a5e76c2a9169373e73f4152ffe3d371dc5 [*] SHA-256 : e8bb0e04a36274f1826530124f54a49cba6b472ec3f39d7a000 5688357be0 [+] File Type: EXE [+] Signature [Compiler/Packer] [*] No match found.</pre>						
	[+] Image Base Address			0000		
	[+] Sections Name: UPX0 Entropy: 6.267019 Name: UPX1 Entropy: 5.194686 Name: .rspc Entropy: 5.096051	Virtual	Address:	0×00001000 0×001ab000 0×0026b000	Size: 0x0 Size: 0x0 Size: 0x0	00c0000
	[+] Anomalies Check					
[*] Header Checksum is zero!						
	[+] Following expected	Malware f	PIs are I	Detected		

#### **References:**

- [1]. Practical Malware Analysis, The hand on guide to dissecting malicious software by Michael Sikorski and Andrew Honig forward by Richard Bejtlich, No starch Press.
- [2] Code Breakers Magazine Security and Anti Security. Attack and Defence by Goppit
- [3] Reversing :Secret of reverse engineering by EldadEilam Foreword by Elliot Chikofsky
- [4] www.securityExploaded.com
- [5] Security Power Tools By: Bryan Burns; Dave Killion. Publisher: O'Reilly Media, Inc



ASHA DEVI Received B.Tech Degree from PDM college of Engineering, Bahadurgarh, INDAI in 2008 and pursuing M.Tech from ITM University Gurgaon, INDIA. I worked as Assistant profeesor in KIIT Gurgaon, INDIA for 4 year and currently working as Independent Security researcher in SecurityExploaded..

#### Acknowledgement:

The authors would like to express their cordial thanks to Mr .Amit Malik(Applied Research, Security - Computer Science) Organization: FireEye) for his valuable advice. LinkedIn : http://www.linkedin.com/in/doublezer0 Website : http://www.securityxploded.com

http://www.securityphresh.com