



REPORT ON UNPACKING OF BOOM_WIN_PE_PACKED.EXE

Ali Abdelhamid El Fekki
Aa2100274@tkh.edu.eg

Table of Contents

Introduction:	2
Methodology and Tools:	2
Initial Identification of Packing Method	2
1. Analysis with PEStudio:	2
2. Entropy Check with DiE (Detect it Easy):	3
3. Further Analysis with PEStudio:	3
Manual Unpacking Process with x64dbg	3
Confirmation of Unpacking	4
Conclusion	5

Introduction:

This report provides a comprehensive description of the manual unpacking process for the executable file named 'boom_win_pe_packed.exe', which was suspected to be packed using UPX. The process documented here outlines the identification of the packing method and the subsequent steps taken to unpack the file manually without relying on the automated UPX tool.

Methodology and Tools:

The unpacking process utilized several tools known for their efficacy in reverse engineering and debugging executables. PESTudio was initially used to inspect the file, suggesting a UPX packing signature. DiE (Detect it Easy) provided a secondary confirmation of the UPX packer. The primary debugging tool used was x64dbg, which facilitated a dynamic analysis of the executable. The Scylla plugin within x64dbg was crucial in reconstructing the Import Address Table (IAT) and finalizing the unpacking process.

Initial Identification of Packing Method

1. **Analysis with PESTudio:** Initial examination of the executable was conducted using PESTudio. This tool provided an overview of the file structure and indicated signs typical of a UPX packed file, such as:

- Writable sections named **UPX0**.
- An entry point is located at **413D75**.
- No imports or exports, which is common in packed executables to evade analysis.
- An overlay size of **27785 bytes**, which can sometimes contain additional packed data.

f:\boom_win_pe_packed.exe	indicator (23)	detail	level
indicators (sections > writable > name)	sections > writable > name	UPX0	1
footprints (count > 6)	entry-point > location	0x0902A093	1
virustotal (unknown)	sections > executable > count	2	1
> dos-header (size > 64 bytes)	sections > self-modifying > name	UPX0 + UPX1	1
> dos-stub (size > 64 bytes)	libraries > count	0	1
> rich-header (n/a)	imports > count	0	1
> file-header (executable > 64-bit)	overlay > size	27785 bytes	2
> optional-header (subsystem > console)	overlay > entropy	4.007	2
directories (count > 3)	file > checksum	0x00000000	2
> sections (characteristics > self-modifying)	sections > virtualized	UPX0	2
libraries (n/a)	groups > API	execution, dynamic-library, memory, file, exception, diagnostic, reconn...	2
imports (n/a)	mitre > technique	T1045, T1106, T1055, T1497, T1124, T1057	2
exports (n/a)	file > entropy	5.512	3
thread-local-storage (count > 1)	file > footprint	4AB13F66C9BB805404D669E901C199A4A4899E6F230D19C8D3A4F3AA71...	3
.NET (n/a)	file > size	40585 bytes	3
resources (n/a)	file > score > error	The requested resource is not among the finished, queued or pending s...	3
abc strings (count > 1482)	file > tooling	MinGW	3
debug (n/a)	security > protection	data-execution-prevention (DEP) > OFF	3
manifest (n/a)	security > protection	control-flow-guard (CFG) > OFF	3
version (n/a)	security > protection	address-space-layout-randomization (ASLR) > OFF	3
certificate (n/a)	file > subsystem	console	3
overlay (signature > MinGW)	thread-local-storage > callbacks	1	3
	security > protection	code-integrity (CI) > OFF	3

2. Entropy Check with DiE (Detect it Easy):

Entropy analysis was performed using DiE, which showed a value of **4.794**. High entropy is indicative of compressed or encrypted data, reinforcing the suspicion that the executable was packed.

3. Further Analysis with PESTudio: The executable was scanned again with PESTudio, revealing:

- The absence of libraries typically used by unpacked executables.
- A total of two executable sections, with self-modifying code present in **UPX0** and **UPX1**.
- The lack of direct imports, which is a characteristic of a UPX packed file.

Manual Unpacking Process with x64dbg

1. **Loading the Executable:** The file "boom_win_pe_packed.exe" was loaded into x64dbg for debugging.

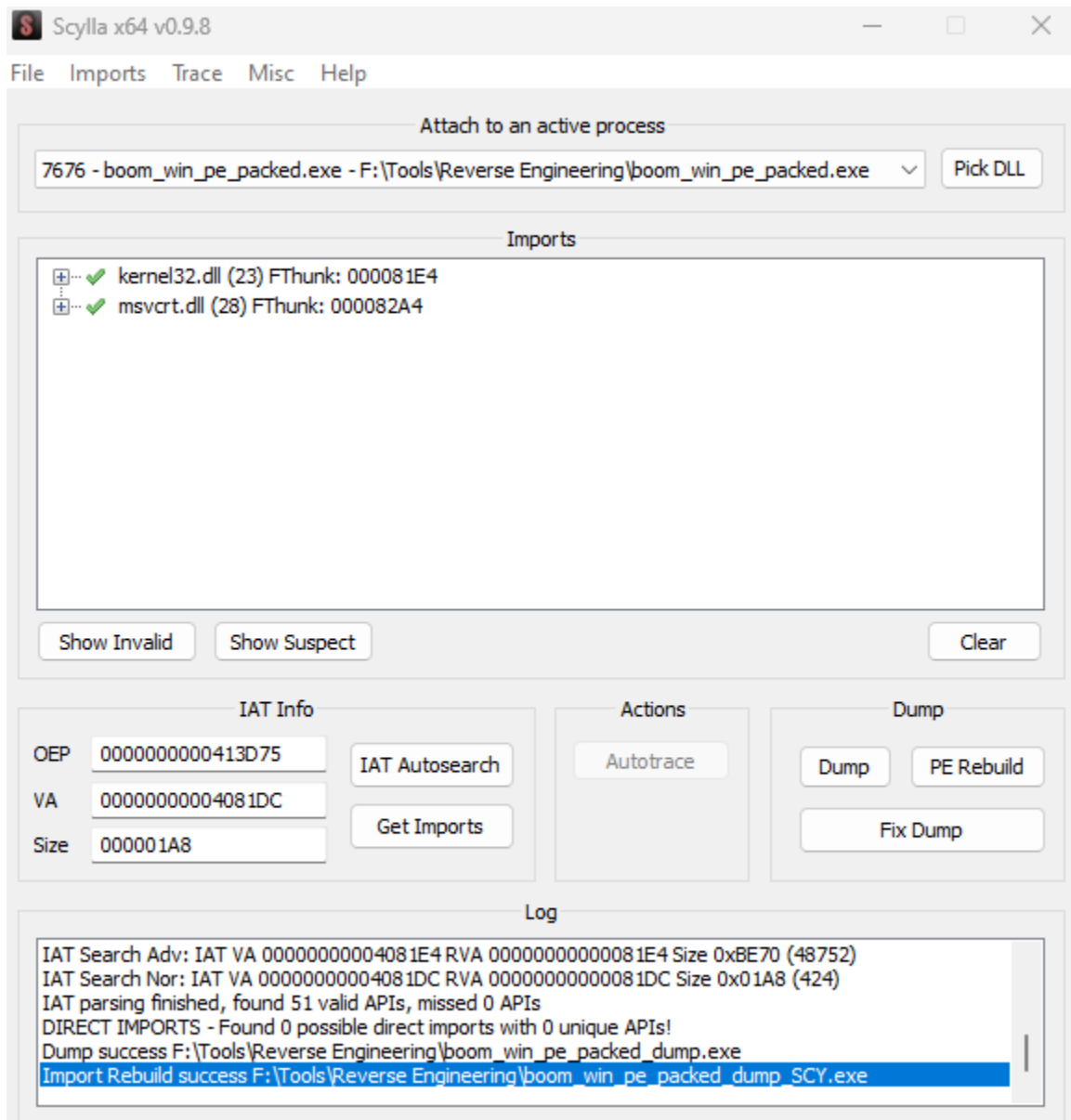
2. **Identifying the Entry Point:** The entry point was identified as a critical location to set a breakpoint and initiate the debugging process.

0000000000413D5B	E8 1A000000	call boom_win_pe_packed.413D7A
0000000000413D60	58	pop rax
0000000000413D61	5D	pop rbp
0000000000413D62	5F	pop rdi
0000000000413D63	5E	pop rsi
0000000000413D64	5B	pop rbx
0000000000413D65	48:8D4424 80	lea rax,qword ptr ss:[rsp-80]
0000000000413D6A	6A 00	push 0
0000000000413D6C	48:39C4	cmp rsp,rax
0000000000413D6F	75 F9	jne boom_win_pe_packed.413D6A
0000000000413D71	48:83EC 80	sub rsp,FFFFFFFFFFFFFF80
0000000000413D75	E9 66D7FEFF	jmp boom_win_pe_packed.4014E0
0000000000413D7A	FC	cld
0000000000413D7B	56	push rsi

3. **Setting a Breakpoint and Running the Executable:** A breakpoint was set at the identified entry point, and the executable was run until it hit the breakpoint.

4. Using Scylla Plugin for IAT Reconstruction:

- The Scylla plugin within x64dbg was utilized to locate the Original Entry Point (OEP).
- An IAT autosearch was conducted to rebuild the Import Address Table (IAT).



- Any suspicious imports marked by Scylla were removed to clean the IAT.
5. **Dumping the Process Memory:** With the correct OEP identified and the IAT cleaned, the process memory was dumped to create a new executable file.
 6. **Repairing the Dumped File:** Scylla's 'Fix Dump' feature was used to repair the newly dumped file, resulting in a restored and functional executable named "boom_win_pe_packed_dump_SCY.exe".

Confirmation of Unpacking

1. **Comparing File Sizes:** The original packed executable, the dumped file, and the repaired file were compared. The increase in file size from the original to the repaired file confirmed the expansion post-unpacking.

2. **SHA-256 Hash Calculation:** The SHA-256 hash of the unpacked file was calculated to verify its integrity and confirm that the file contents had been altered through the unpacking process.
3. **Final Verification with PESTudio:** The fully unpacked executable was scanned one last time with PESTudio to ensure all signs of UPX packing had been removed, confirming the success of the manual unpacking process.

The screenshot shows the PESTudio interface with the 'Entropy' tab selected. The 'Regions' table lists the following data:

Offset	Size	Entropy	Status	Name
00000000	00000400	1.55662	not packed	PE Header
00000400	0000f200	2.65568	not packed	Section(0)['UPX0']
0000f600	00002e00	5.48204	not packed	Section(1)['UPX1']
00012400	00000200	1.96753	not packed	Section(2)['UPX2']
00012600	00000600	4.18633	not packed	Section(3)['.SCY']

Conclusion

The boom_win_pe_packed.exe file was conclusively identified as a UPX-packed executable using PESTudio. This was confirmed by several indicators, including specific section names, high entropy values, and an empty import table. Through the detailed steps outlined above, "boom_win_pe_packed.exe" was successfully unpacked manually. This process required a deep understanding of the executable structure, UPX packing mechanisms, and the use of advanced debugging tools to reverse-engineer the file to its original state.