Intro to Software Protection

CrackingLessons.com

Contents

- What is Software Protection?
- What is EXE Packing?
- Purpose of Packing EXE
- How to defeat Software Protection
- What is Unpacking?
- Detection of Packer
- Execution of Packed EXE Program
- Standard Process of Unpacking EXE
- Anti Anti-Debugging Plugins

What is Software Protection?

- Protection of software against piracy, overuse, and reverse engineering

Ways to protect software:

- Anti debugging (Prevent debuggers from attaching or analyzing)
- Exe Packing (Compress a software whilst retaining its ability to execute)

What is EXE Packing/Protecting?

- **EXE Packing:**

  Compressing the Executable to a smaller Size

- **EXE Protecting:**

  Using Anti-Debugging Techniques to prevent Reversing

In Reversing world, both Packer & Protector is commonly referred as **Packer**.

Examples of Packers: UPX, AsProtect, Armadillo, VMProtect etc.

Purpose of Packing EXE

- Prevent Reverse Engineering [Crack License, Serial Key etc.]
  - Defeat Static Disassembling
  - Make Dynamic Debugging Difficult
- Reduce the size of Executable file

How to defeat Software Protection?

Unpacking:

Let the program uncompress itself into memory, then extract the original exe from memory and dump it into a new exe file. Then patch the new exe.

Using Loaders:

This is also known as runtime patching. Here, you patch the process in memory instead of patching the file. Use a loader to start the program and wait for it to uncompress itself into memory. Loader will then patch the process whilst it is still running in memory.

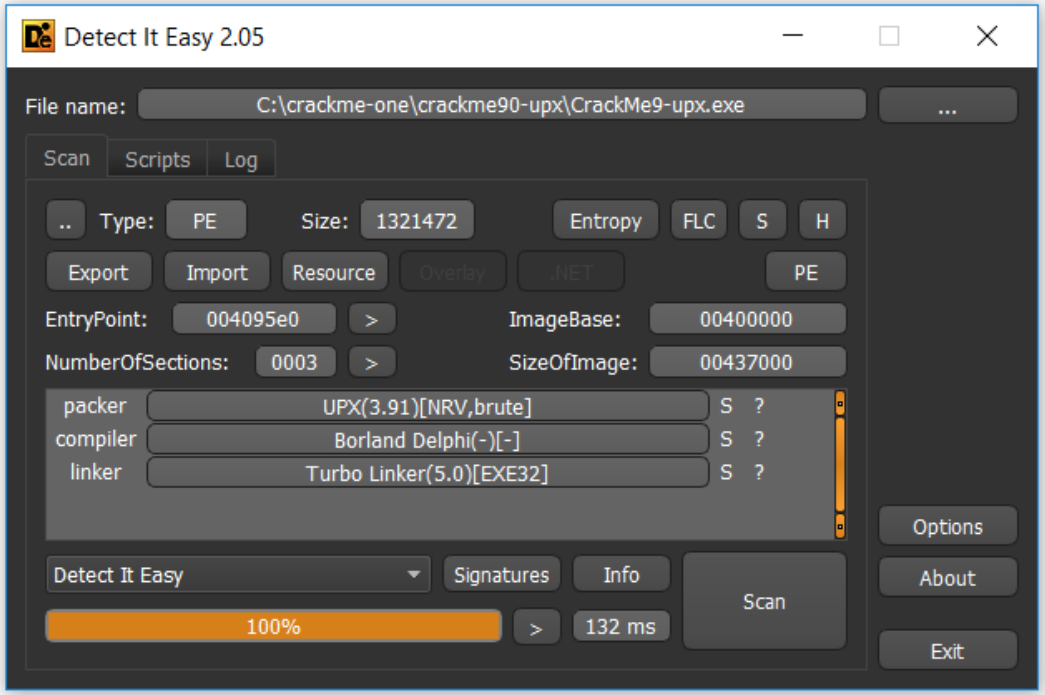
What is Unpacking?

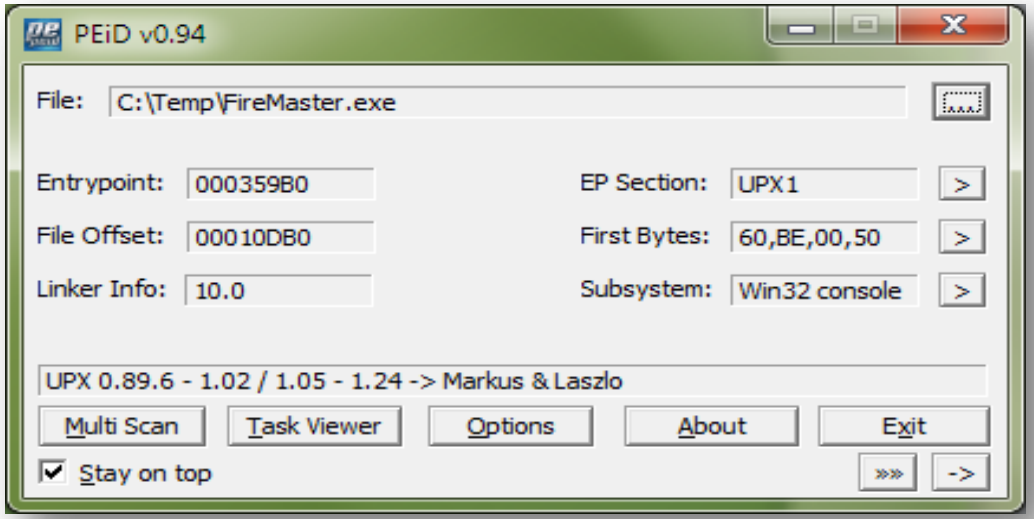
- Extracting the Original Binary from the Packed Executable File.
- Automatic Unpackers available for popular Packers.
  - May not work with different versions
  - Not available for Complex packers
- Involves Live Debugging by Defeating Anti-Debugging techniques

Detection of Packer

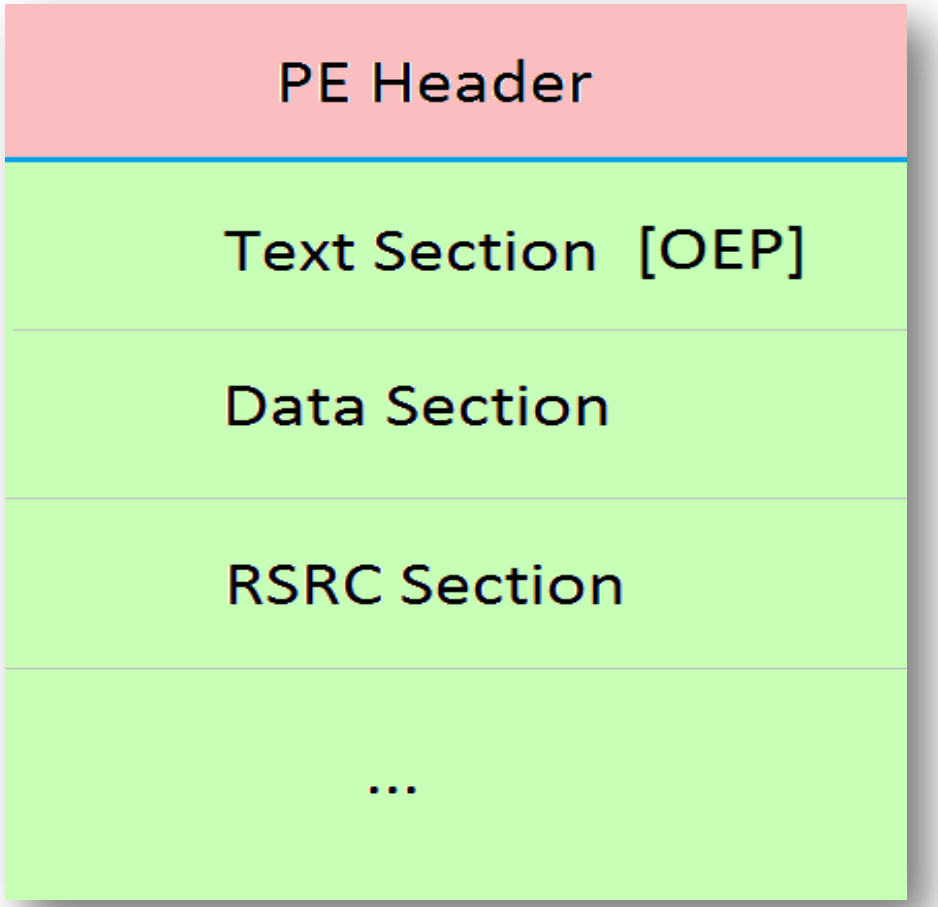
- Packer Detectors like PEiD, DIE, etc
  - Detect the popular Packers
  - Show the version of Packer also
- PE Viewer Tools like PEditor, PEview
  - Look at Section Table
  - Look at Import Table

Packer Detectors

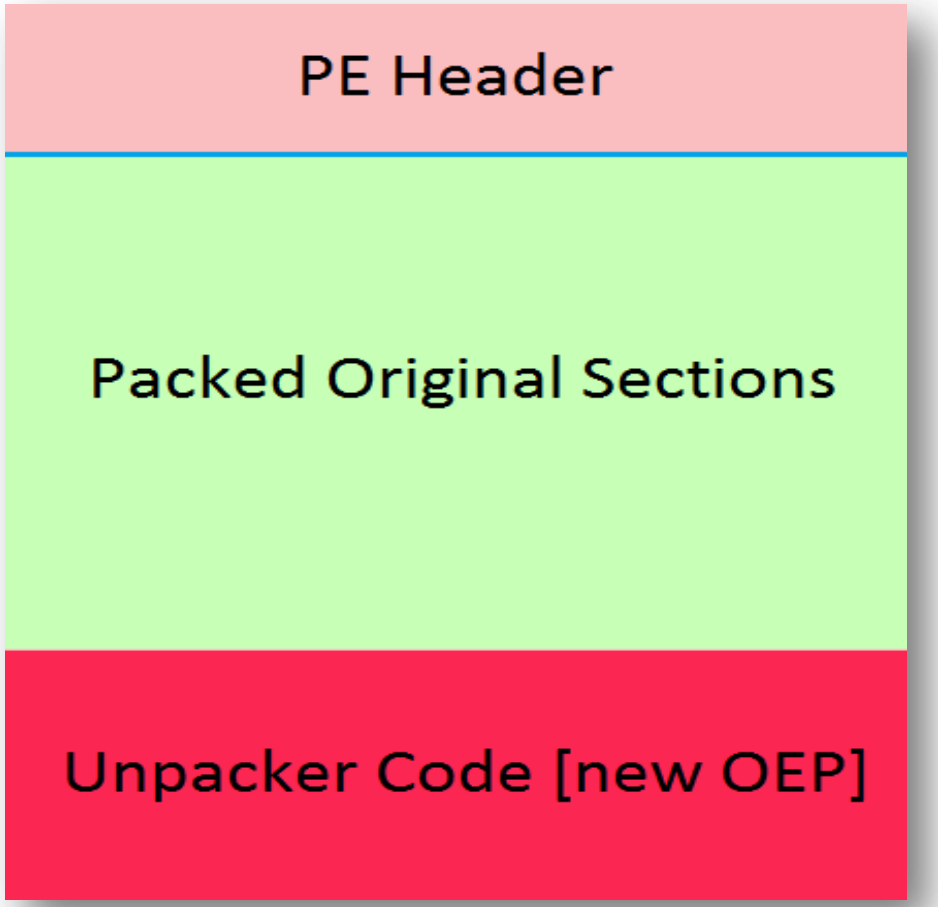




Structure of Packed EXE



Before Packing



After Packing

Types of Packers

Type I
The simplest type of packer that employs a single unpacking routine being executed before control transferred to the unpacked program. An example is UPX.

Type II
It contains multiple unpacking layers, individually executed sequentially to reveal the subsequent routine. Once reconstruction of the original code is completed, the control is transferred back to it at the last transition.

Type III
Similar to the previous type of packer, the variation lies in that the execution of the unpacking routines are not in a straight line, involving more complex structures, such as loops. The original code may not be necessarily resided in the last (deepest) layer. Different codes like integrity checks, anti-debugging, or portioning of the obfuscated code of the packer may be found in the last layer. Examples include PE Compact, ASPack, FSG, ASProtect, NSPack, and Upack.

Types of Packers

Type IV

This type of packer is either single- or multilayer packer, in which a portion of the packer code, not responsible for unpacking, is interleaved with the execution of the original program. The complete unpacking of the entire original code can be found in the memory, with the possibility of code jumping between different layers at the the Final execution. An Example is ACProtect 1.09.

Type V

An interleaved packer, with the unpacking code is inter-weaved with the original program. Multiple frames are usually found in thelayer embodied in the original code, being unpacked one after another. Although only a single frame of code is unfolded, all the executed code may be extracted at the end of execution. An example this type is Beria.

Types of Packers

Type VI
The most complex type of packer, this type unpacks fragments of code at any given time during the execution of the binary code. An example is Armadillo 8.0.

Type VII
Virtualization packers use instruction translation to avoid the original code being exposed in the memory at all.
Examples are Themida, VMProtect.

Execution of Packed EXE Program

- Execution starts from new OEP (Original Entry Point)
- Saves the Register status using PUSHAD or PUSH EBP instruction
- All the Packed Sections are Unpacked in memory
- Resolve the import address table (IAT) of original executable file.
- Restore the original Register Status using POPAD or POP EBP instruction
- Finally Jumps to OEP to begin the actual execution

Execution of Packed EXE Program

- A single PUSHAD instruction is equivalent to:

- Push EAX
- Push ECX
- Push EDX
- Push EBX
- Push ESP
- Push EBP
- Push ESI
- Push EDI

Execution of Packed EXE Program

- A single POPAD instruction is equivalent to:

- Pop EAX
- Pop ECX
- Pop EDX
- Pop EBX
- Pop ESP
- Pop EBP
- Pop ESI
- Pop EDI

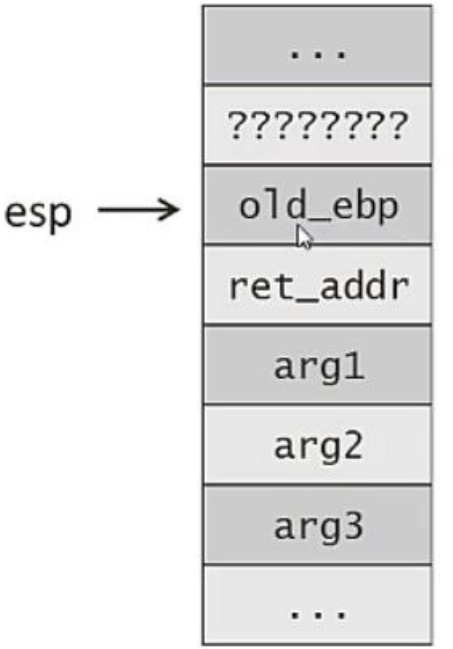
Holding the stack frame

- We could use EBP to remember what ESP was initially.
  - Then we don't care about changes to esp.

```
func:
    push    ebp ; Keep ebp
    mov     ebp,esp

    ...

    pop     ebp ; Restore ebp
    ret
```



Standard Process of Unpacking EXE

- Debug the EXE to find the real OEP (Original Entry Point)
- At OEP, Dump the fully Unpacked Program to Disk
- Fix the Import Table
- Fix the PE Header

Unpacking using x64dbg

- Load the packed EXE file into the x64dbg
- Start tracing the EXE, until you encounter a PUSHAD, or PUSH EBP instruction.
- Put a Hardware Breakpoint on the EBP address in the stack

Unpacking UPX using x64dbg (contd)

- Next press F9 to continue the Execution.
- You will break on the instruction which is immediately after POPAD or on POP EBP instruction [based on the method you have chosen]
- Now start tracing with F8 and soon you will encounter a JMP instruction which will Jump to OEP in the original program.
- At OEP, dump the whole program using x64dbg Scylla Plugin
- Then fix the IAT

Anti Anti-Debugging Plugins

Here are useful x64dbg Plugins for Anti Anti-Debugging

- ScyllaHide
- SharpOD

Thank you

CrackingLessons.com