4 Making a Multi-Windows PE

by Ange Albertini

4.1 Evolution of the PE Loader

The loader for PE, Microsoft's *Portable Executable* format, evolved slowly, and became progressively stricter in its interpretation of the format. Many oddities that worked in the past were killed in subsequent loader versions; for example, the notorious TinyPE[4] doesn't work after Windows XP, as subsequent revisions of Windows require that the `OptionalHeader` is not truncated in the file, thus forcing a TinyPE to be padded to 252 bytes (or 268 bytes in 64 bit machines) to still load. Windows 8 also brings a new requirement that $AddressOfEntryPoint \leq SizeOfHeaders$ when $AddressOfEntryPoint \neq 0$, so old-school packers like FSG[5] no longer work.

So there are many real-life examples of binaries that just stop working with the next version of Windows. It is, on the other hand, much harder to create a Windows binary that would continue to run, but differently—and not just because of some explicit version check in the code, but because the loader's interpretation of the format changed over time. This would imply that Windows is not a single evolving OS, but rather a succession of related yet distinct OSes. Although I already did something similar, my previous work was only able to differentiate between XP and the subsequent generations of Windows.[6] In this article I show how to do it beyond XP.

4.2 A Look at PE Relocations

PE relocations have been known to harbor all sorts of weirdness. For example, some MIPS-specific types were supported on x86, Sparc or Alpha. One type appeared and disappeared in Windows 2000.

Typically, PE relocations are limited to a simple role: whenever a binary needs to be relocated, the standard Type 3 (`HIGH_LOW`) relocations are applied by adding the delta $LoadedImageBase - HeaderImageBase$ to each 32 bit immediate.

However, more relocation types are available, and a few of them present interesting behavioral differences between operating system releases that we can use.

Type 9 This one has a very complicated 64-bit formula under Windows 7 (see Roy G Big's `vcode2.txt` from Valhalla Issue 3 at `http://spth.virii.lu/v3/`), while it only modifies 32 bits under XP. Sadly, it's not supported anymore under Windows 8. It is mapped to `MIPS_JMPADDR16`, `IA64_IMM64` and `MACHINE_SPECIFIC_9`.

Type 4 This type is the only one that takes a parameter, which is ignored under versions older than Windows 8. It is mapped to `HIGH_ADJ`.

Type 10 This type is supported by all versions of Windows, but it will still help us. It is mapped to `DIR64`.

So Type 9 relocations are interpreted differently by Windows XP and 7, but they have no effect under Windows 8. On the other hand, Type 4 relocations behave specially under Windows 8. In particular, we can use the Type 4 to turn an unsupported Type 9 into a supported Type 10 only in Windows 8. This is possible because relocations are applied directly in memory, where they can freely modify the subsequent relocation entries!

[4] `http://www.phreedom.org/research/tinype/`
[5] Fast Small Good, by bart/xt
[6] See "TLS AddressOfIndex in an Imports descriptor" for differentiating OS versions by use of Corkami's `tls_aoiOSDET.asm`.

4.3 Implementation

Here's our plan:

1. Give a user-mode PE a kernel-mode `ImageBase`, to force relocations,
2. Add standard relocations for code,
3. Apply a relocation of Type 4 to a subsequent Type 9 relocation entry:
   - Under XP or Win7, the Type 9 relocation will keep its type, with an offset of $0f00h$.
   - Under Win8, the type will be changed to a supported Type 10, and the offset will be changed to $0000h$.
4. We end up with a memory location, that is either:

   XP Modified on 32b ($00004000h$),

   Win7 modified on 64b ($08004000h$), or

   Win8 left unmodifed ($00000000h$), because a completely different location was modified by a Type 10 relocation.

```
;relocation Type 4, to patch unsupported relocation Type~9 (Windows~8)
block_start1:
    .VirtualAddress dd relocbase - IMAGEBASE
    .SizeOfBlock dd BASE_RELOC_SIZE_OF_BLOCK1

    ; offset +1 to modify the Type, parameter set to -1
    dw (IMAGE_REL_BASED_HIGHADJ << 12) | (reloc4 + 1 - relocbase), -1
BASE_RELOC_SIZE_OF_BLOCK1 equ - block_start1

; our Type 9 / Type 10 relocation block:
; Type 10 under Windows8,
; Type 9 under XP/W7, where it behaves differently
block_start2:
    .VirtualAddress dd relocbase - IMAGEBASE
    .SizeOfBlock dd BASE_RELOC_SIZE_OF_BLOCK2

; 9d00h will turn into 9f00h or a000h
reloc4 dw (IMAGE_REL_BASED_MIPS_JMPADDR16 << 12) | 0d00h
BASE_RELOC_SIZE_OF_BLOCK2 equ $ - block_start2
```

We now have a memory location modified transparently by the loader, with a different value depending on the OS version. This can be extended to generate different code, but that is left as an exercise for the reader.