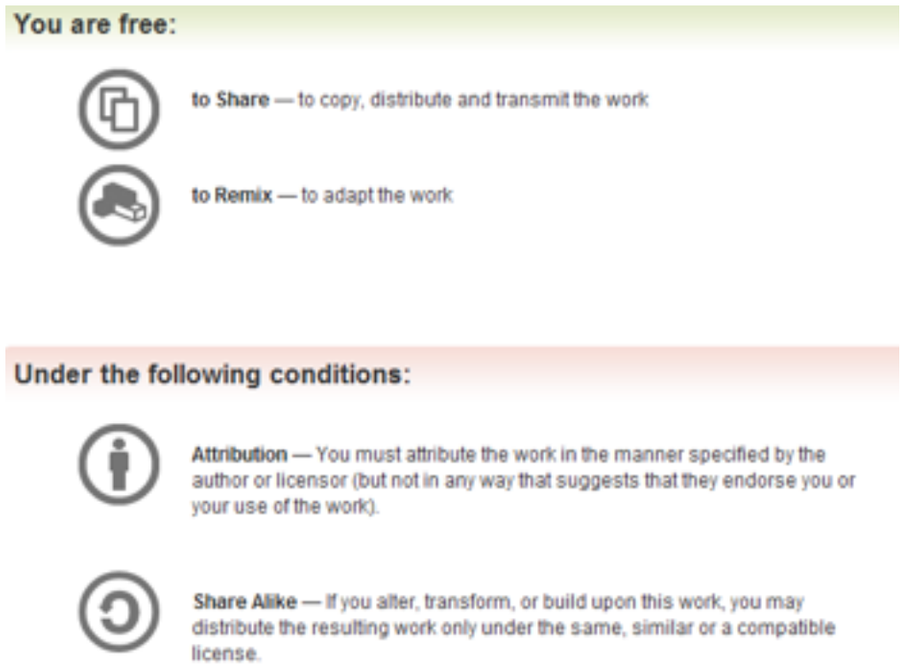
Introduction to Intel x86-64 Assembly, Architecture, Applications, & Alliteration

Xeno Kovah – 2014-2015
xeno@legbacore.com

All materials is licensed under a Creative Commons “Share Alike” license.

- http://creativecommons.org/licenses/by-sa/3.0/



Attribution condition: You must indicate that derivative work
"Is derived from Xeno Kovah's 'Intro x86-64' class, available at http://OpenSecurityTraining.info/IntroX86-64.html"

Attribution condition: You must indicate that derivative work

"Is derived from Xeno Kovah's ‘Intro x86-64’ class, available at http://OpenSecurityTraining.info/IntroX86-64.html"

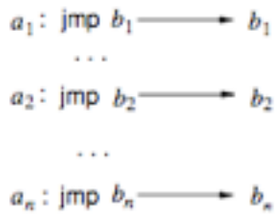
Messing with a disassembler

- Obfuscation of Executable Code to Improve Resistance to Static Disassembly - Linn & Debray
  - http://www.cs.arizona.edu/solar/papers/CCS2003.pdf
  - Linear sweep vs. recursive traversal disassembly
  - Also discusses and measures the “self-repairing” nature of x86 disassembly which we saw earlier
- Confusing linear sweep (objdump) by inserting junk bytes after unconditional jumps.
  - Could be literally unconditional “jmp”
  - Could be a jcc, which must always be true, like “xor eax, eax” and then “jz <addr>”
  - Have to do this multiple times because of the self-repairing disassembly

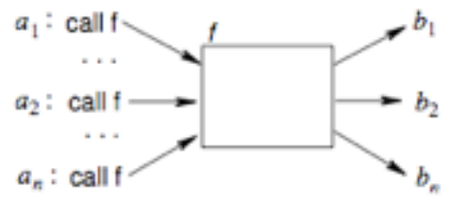
Messing with disassembler 2

- Confusing recursive traversal
  - 3.4.1: Branch functions. All jmps turned into a call to a specific function.
  - 3.4.2: Call conversion. Branch functions + the junk byte technique which messed with linear sweep.
  - 3.4.3: Opaque predicates. Create ostensibly conditional jumps which will in fact always follow only one path. The disassembler doesn’t have the smarts to determine this.
  - 3.4.5: Jump table spoofing. Exploits the fact that the disassembler may try to estimate the size of the jump table based on a constraint. The trick is to add a jump table which will never be reached.

Branch Functions Visualized



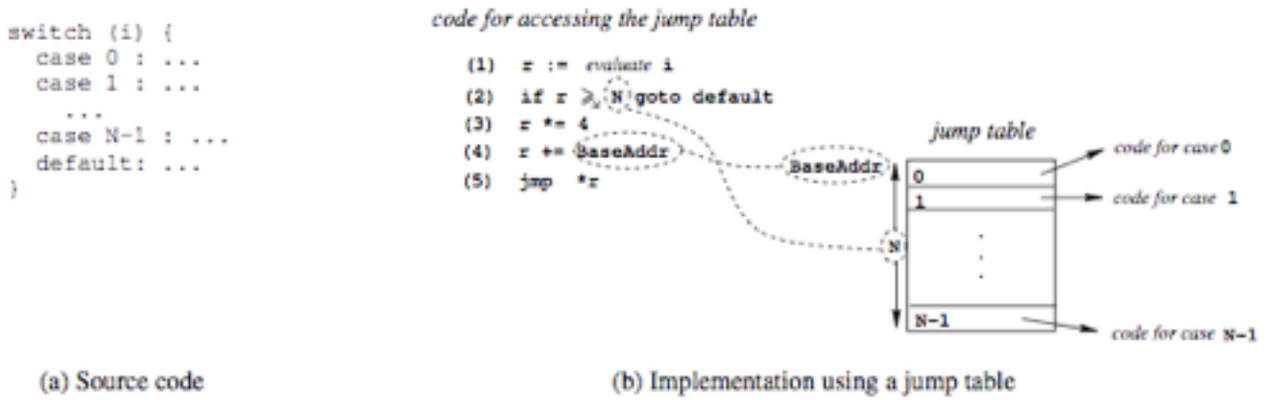
(a) Original code



(b) Code using a branch function

Figure 5: Branch functions

Jump table visualized



(a) Source code

(b) Implementation using a jump table

Figure 3: A example of a C `switch` statement and its implementation using a jump table

Actual implementation of many of these techniques

- Nick Harbour actually apparently independently came to the same conclusion as Linn & Debray a few years later, and made a tool that performed some of these obfuscations, and did a Defcon talk about it
  - https://www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-harbour.pdf
  - http://www.youtube.com/watch?v=wdFLK_eX0QY
  - https://web.archive.org/web/20100324144525/http://www.rnicrosoft.net/tools/PEScrambler_v0_1.zip

Addressing Linn & Debray obfuscations

- Static Disassembly of Obfuscated Binaries - Kruegel *et al.*
  - http://www.cs.ucsb.edu/~chris/research/doc/usenix04_disasm.pdf
  - Attempt to improve on the state of the art in disassembling, to deal with the Linn & Debray obfuscations
  - I don't know if there are any disassemblers which try to use these improved disassembly methods (objdump and IDA definitely don't). Confirmed with Kruegel that he's not aware of anywhere that uses the improvements either.