

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

Xeno Kovah – 2014
xkovah at gmail

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

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

You are free:



to **Share** — to copy, distribute and transmit the work



to **Remix** — to adapt the work

Under the following conditions:



Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

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>"

Bomb lab

- From CMU architecture class - <http://csapp.cs.cmu.edu/public/labs.html>
- Thanks to Randal E. Bryant & David R. O'Hallaron for providing the source code so it could be ported to x86-64 (and Windows in the Intro RE class)
- The textbook for the class which the bomb lab is a part of is "[Computer Systems: A Programmer's Perspective, 2nd Edition, Prentice Hall, 2011; Bryant and O'Hallaron](#)"

Bomb lab 2

- Goal is to reverse engineer multiple phases to determine the program's desired input
- Create a text file with answers, one per line, named "answers"
- `gdb -x myCmds bomb`
- run with "r < answers"
- Should add/remove breakpoints on the different phases as you go along

Bomb lab - EXPERT MODE!

- If you already know a thing or two about asm (and were just here for the 64 bit update), let's see how far you can get how fast if you play it on expert mode, without symbol information. Execute the following command in the directory where the bomb resides:
 - strip bomb
 - This is more like what you will actually see with malware. You're not going to get symbols in that case.
 - *Now go ahead and see how fast you can go through the rounds ;)*

Phase_2 hint

- `sscanf()` is defined as follows:
- `int sscanf(const char *str, const char *format, ...);`
- So if it was e.g. "`sscanf(foo, "%d %d", &a, &b)`"
- It would take whatever string was pointed to by the first argument, *parse* it according to the second format string argument, and then store the parsed out values in the variables which were given by the subsequent `n` arguments (for `n = 2` in this case)
- "On success, the function returns the number of variables filled"

Phase_2 hint

- `sscanf()` is defined as follows:
- `int sscanf(const char *str, const char *format, ...);`
- So if it was e.g. “`sscanf(guess, “%d %d %d %d %d %d”, &var1, &var2, &var3, &var4, &var5, &var6,)`”

GDB/Bomb Lab Cheat Sheet

- Christian Arllen found this, and it has many more example of gdb syntax, as well as some help for if you get stuck on the lab
- <http://condor.depaul.edu/~jriely/csc373fall2010/extras/mygdbnotes.txt>
- (get it on google cache while you can, because it's gone now)