Lab: RE bingo

Environment Needed:

- Linux Virtual Machine (Recommend Ubuntu)

Links for any tools are available in the github under 'Tools'

Introduction

- In this lab, we will examine numerous programs to determine what control flow and compiler settings each uses. Correctly analyzing all programs will allow us to open the lockbox.zip at the end of each challenge.
- Each program contains three functions: main(), key_check(), and do_stuff(). main() calls key_check, key_check() calls do_stuff, and do_stuff() performs several no-ops ("nop" instructions). key_check() contains the control flow to analyze.
- Five types of control flow are used: if statements, if-else statements, do-while loops, switch statements, and for-loops. Note that there are no while loops, only do-while loops. In each key_check() function uses exactly one control flow construct.
- Each program is compiled with optimization or no optimization, and with symbols or stripped symbols.
- As always, the primary goal is to complete part I of the lab. If you solve part I, tackle part II for a more in depth challenge and practice.

1. Use objdump to examine the first program in the part_i directory (the program name '1'). Determine what control flow construct is being used by key_check, whether the program is optimized or not optimized, and whether symbols are stripped or not stripped.
2. Write down your determination as follows:
    a. "i" for an if statement,
    b. "e" for an if-else statement,
    c. "w" for a do-while loop,
    d. "s" for a switch statement,
    e. "f" for a for loop,
    f. then append "x" if the program symbols are stripped,

X86-SOFTWARE-REVERSE-ENGINEERING-CRACKING-AND-COUNTER-MEASURES
ISBN: **1394199880**

   g. then append "o" if the program was compiled with optimizations.
3. For example, if the first program is an optimized, stripped for loop, write"fxo". If it is an unoptimized, unstripped switch statement, write "s". Repeat this process for all 8 programs, in order.
4. When you are confident in your answers, append each of your solutions, in order, separated by and underscore _. For example, if you decided that all eight problems are optimized, unstripped do-while loops, your solution would be: wo_wo_wo_wo_wo_wo_wo_wo
5. Use this key as the password to unlock the lockbox: unzip lockbox.zip
   a. If your solution is correct, the lockbox will open and reveal your reward.
   b. If the solution is incorrect, unzip will inform you that the password is wrong - re-evaluate the problems and carefully investigate any area you are cunsure about.

== Challenge: Part II ==

1. Part II is similar to part I. Navigate to the part_ii directory. This time there are far more challenges to solve. Often in advanced cracking, it is necessary to write scripts to automate certain reverse engineering tasks that would be too time consuming to complete by hand.
2. This challenge uses the same options for control flow and symbol stripping, but does not use compiler optimization.
3. In a scripting language of your choice, _automate_ the identification of control flow construct and symbol setting for each of the part_ii challenge problems in order, to automatically derive the key and unlock the advanced lockbox: unzip lockbox.zip
4. Note: if you are not familiar with any scripting language in Linux, that's okay! This problem _can_ be solved by hand, and will be good practice when we get to more advanced constructs moving forward.

X86-SOFTWARE-REVERSE-ENGINEERING-CRACKING-AND-COUNTER-MEASURES
ISBN: **1394199880**