Exercise 2 – Smashing the stack for fun and profit

README.1ST:

1. Before starting the exercise:
    a. Run “ex_unpack upd.bin upd”
    b. This will update the updater! Make sure there are no errors, if there are please contact us immediately.
    c. Now run “ex_unpack ex02.bin ex02” – this will open the exercise
    d. Enter the exercise directory ex02
    e. Use ls –lrt. You will notice some of the files have the sticky bit!
    f. Now lets get to it!

Stack overflows!

- IMPORTANT NOTE, in this section, the binaries do not match the exact source code on purpose.
    - The passwords and the way the are stored has been changed from the source code, to make things a bit more challenging for you.

2. Read the off_by_one.c code. Can you make the password check succeed without brute forcing the password? [explain in your own words in a txt what you did. submit as 'ex2_2.txt']
    a. Look for a bad boundary check on a buffer.
    b. Use IDA to view the stack and find the buffer.
    c. Look and see what will be overwritten due to the boundary check failure.
    d. Modify the input so it will overwrite it properly so that it can work. (Use python to invoke the program with different parameters, submit as 'ex2_2d.py').
    e. Use the program to run “sh” [Write in the summary which parameters you used]
    f. Write a python script to crack the password. [submit as 'ex2_2f.py']
3. Preparations for question 4.
    a. **Now that you are root**
    b. Use the new achieved shell to vi /etc/group
    c. Look for the lines where the word instructor appears at the end and add “,student”

d. Example:
    i. sudo:x:27:instructor
    ii. turns to
    iii. sudo:x:27:instructor,student
e. You will now have sudo capability!
f. Logout and login.
g. Make sure you can sudo to root by doing “sudo su” – a root shell should appear.
h. Go back to user mode.
i. Go to the main exercise library run “sudo ./tools/exercise_mode.sh” – this will disable ASLR defenses and enable core files so you can debug your errors!
j. Now **restart** your VM.

4. Running arbitrary code:
[Please write down a README of what you did, and why as a full answer for this question. Submit as 'ex2_4.txt']
Look at stacko1.c - this code checks for passwords and logs commands from a file into a file:
For every line in the file if the password fits it echos the command supplied into a log file owned by root.
    a. Use the command password achieved in question 2! Use it to run the program and echo the commands to a file, as the original function.
    b. Find another bad boundary check. (Hint: this time, your input sets the limit).
    c. Use IDA to see how far you can override to reach the return pointer.
    d. Use python to generate a file with bad input and force the process to core dump [submit as 'ex2_4.py']
    e. Use sudo gdb --core=tmp.core to load the core file.
        i. Use: dump memory [filename] [stack_start] [stack_stop].
        ii. Use ghex and find the beginning of the buffer in the memory
    f. Add shellcode and jump to it!
        i. “\xeb\x0f[1\xc0\xb0\x0b1\xc9\xcd\x801\xc0\xb0\x01\xcd\x80\xe8\xec\xff\xff\xff/bin/sh” /* please note unprintable characters are in hex format, and printable are shown as is. - this is good for

use with python, for other languages make sure it is interpreted properly. */

g. Pad the shellcode with a NOP slide (“\x90”). And try jumping
h. Your payload is now a shellcode. I’m on a horse.
i. BONUS: Run sh without overflowing a buffer [10pts]