**CSC 373 Sections 901, 910 Spring 2018**

**Final Exam**

**Name:** Logan Murray

**Directions:** This is a take-home exam, due on June 8. Please write your answers in this file, and then upload your answers to the submissions box by the 8th..The exam is open book and open notes. You may use computing devices as well. You must work alone; specifically, you may not communicate with anyone else (live or electronically) during the exam

**Exam questions**

**1**.  **(12 points total)** Below is an annotated session with the Gnu debugger running an executable called problem1. As you can see, the executable contains a function called p1.

linux> gdb problem1

(gdb) break \*p1

Breakpoint 1 at 0x400660

(gdb) run

Breakpoint 1, 0x00400660 in p1 ()

(gdb) disas

=> 0x00400660 <+0>: movslq 0x1c(%rdi),%rax

0x00400664 <+4>: movslq %esi,%rsi # ignore this instruction

0x00400667 <+7>: movl $0x3c,0xc(%rdi)

0x0040066e <+14>: movl $0x5a,(%rdi,%rax,4)

0x00400675 <+21>: movl $0x50,-0x4(%rdi,%rsi,4)

0x0040067d <+29>: mov 0xc(%rdi),%eax

0x00400680 <+32>: retq

(gdb) print/x $rdi

$1 = 0x7fffffffe960

(gdb) x/8d $rdi # prints 32 bytes as 8 integers

0x7fffffffe960: 70 60 50 40

0x7fffffffe970: 30 20 10 0

(gdb) print/d $rsi

$1 = 8

A. (2 points) Write a prototype for p1. Remember that a prototype only specifies the return type of a function, the number of parameters that it is passed, and the types of these parameters.

🡪 int[] p1 (int\* , int );

B. (10 points) Fill in the table below. Keep in mind the values in the array that starts at memory location 0x7fffffffe960. If the destination of an instruction is a memory location, write the memory address as a hex number. You may write only the last 4 digits of the address; for example,!0x7fffffffe960 may be written as e960. In your answers, assume the instructions run in sequence; thus, any changes in memory or register contents persist from one instruction to the next.

|  |  |  |  |
| --- | --- | --- | --- |
| Line | Destination | Old value | New value |
| p1+0 | %rax | - | 0 |
| p1+7 | 0x..e96c | 40 | 0x3c |
| p1+14 | 0x..e960 | 70 | 0x5a |
| p1+21 | 0x..e97c | 0 | 0x50 |
| p1+29 | %eax | 0 | 0x3c |

**2. (3 points)** Complete the getbit function below. It returns the value of the ith bit of an integer (bit 0 is the rightmost bit).

int getbit(int x, int i) {

if (x & (1 << (i - 1))) {

return 1;

} else {

return 0;

}

}

**3. (5 points total)** Consider the assembly language function p3 below. Note that jne is used somewhat differently than we have seen in the past, in that it is not immediately preceded by a comparison. Instead, it is the subl instruction which determines whether or not the jump takes place; specifically, the jump occurs if the result of the subtraction is nonzero.

(gdb) disas p3

Dump of assembler code for function p3:

0x400610 <+0>: test %esi,%esi

0x400612 <+2>: je 0x400623 <p3+19>

0x400614 <+4>: nopl 0x0(%rax) # nopl ("no-op") does nothing and is for alignment

0x400618 <+8>: mov %esi,(%rdi)

0x40061a <+10>: add $0x4,%rdi

0x40061e <+14>: sub $0x1,%esi

0x400621 <+17>: jne 0x400618 <p2+8>

0x400623 <+19>: retq

End of assembler dump.

The above assembly language was the result of compiling a C function of the format below. Your job is to fill in the missing pieces of the C code, so that it emulates p3. You will fill in 3 portions, each part is worth 1 point. The missing pieces are indicated by the dots below. Replace the dots and complete the C code.

void p3(int \*x, int len) {

int i;

for (i=len; i > 0; i--)

{\*x = i; }

}

**4. (5 points total).** Consider the assembly language function below. I have added some comments to help you understand the code.

(gdb) disas p4

Dump of assembler code for function p4:

0x4005c0 <+0>: movzbl (%rdi),%edx

0x4005c3 <+3>: test %dl,%dl # %dl is the least significant byte of %rdx

0x4005c5 <+5>: je 0x4005e9 <p4+41>

0x4005c7 <+7>: add $0x1,%rdi

0x4005cb <+11>: xor %eax,%eax

0x4005cd <+13>: nopl (%rax) # nopl ("no-op") does nothing and is for alignment

0x4005d0 <+16>: cmp %sil,%dl # %sil is the least significant byte of %rsi

0x4005d3 <+19>: sete %dl

0x4005d6 <+22>: add $0x1,%rdi

0x4005da <+26>: movzbl %dl,%edx

0x4005dd <+29>: add %edx,%eax

0x4005df <+31>: movzbl -0x1(%rdi),%edx

0x4005e3 <+35>: test %dl,%dl

0x4005e5 <+37>: jne 0x4005d0 <p4+16>

0x4005e7 <+39>: retq

0x4005e9 <+41>: xor %eax,%eax

0x4005eb <+43>: retq

1. (1 point)The first parameter of p4 is an array. What type of data does the array contain? Explain your answer.

* On lines <+7> and <+22> the memory address of the start of the array is only being increased by 0x1, which my understanding means by just 4 bits or half a byte. Which makes no sense to me seen as no datatype is half a byte. My best guess is that the array is full of type “char.”

1. (1 point) What type of data is passed as the second parameter of p4? Explain.

🡪 I think it would also be just a “char” type because it only uses the least significant byte (%sil) when it runs a comparison between what is in %edx and %esi. But I have no idea what any of this means so it could be anything.

C. (1 point) If a function is passed a third parameter, it is placed in register %rdx..

Note that %rdx (!%dl!) is used in line <p4+3>,, among other places. Is this function passed a third parameter? Explain your answer.

* I believe there is not a third parameter, but within the function there might be a variable being set equal to the first value of the array, seen as the first line is moving the memory address stored in rdi (the array) into edx.

1. (1 point) What type of data (if any) does the function return?

* I think it is returning whatever it found to be equal in the array. I think it is a search and find function.

1. (1 point) In 25 words or less, describe in English what p4 does.

* It is passed an array of char and a char as parameters and then it scans through the array until it finds what it was looking for and then sets %eax to what it found and then returns after that.

**5 (5 points total)** Below is an interaction using gdb with final\_bomb,

an executable that is similar to the bomb from homework assignment

6. In final\_bomb, there is one phase, called p5.

\footnotesize

\begin{verbatim}

linux> ./final\_bomb

Welcome to the final exam bomb. Type your user id

guest

Try your hand at phase p5...

asodifj

BOOM!!!

The bomb has blown up.

P5 is passed one parameter, containing the user's input (e.g., ``asodifj'').

Its behavior is dependent on the input and the user ID. Here is a disassembly of p5!.

Notice that the user ID is stored in a global variable, whose address is 0x60104c

(gdb) disas p5

Dump of assembler code for function p5:

0x00000000004006b0 <+0>: sub $0x18,%rsp

0x00000000004006b4 <+4>: xor %eax,%eax

0x00000000004006b6 <+6>: mov $0x4007e1,%esi

0x00000000004006bb <+11>: lea 0xc(%rsp),%rdx

0x00000000004006c0 <+16>: callq 0x400520 <\_\_isoc99\_sscanf@plt>

0x00000000004006c5 <+21>: xor %eax,%eax

0x00000000004006c7 <+23>: cmpb $0x0,0x20097e(%rip) # 0x60104c <user\_id>

0x00000000004006ce <+30>: je 0x4006df <p4+47>

0x00000000004006d0 <+32>: add $0x1,%eax

0x00000000004006d3 <+35>: movslq %eax,%rdx

0x00000000004006d6 <+38>: cmpb $0x0,0x60104c(%rdx)

0x00000000004006dd <+45>: jne 0x4006d0 <p5+32>

0x00000000004006df <+47>: cmp 0xc(%rsp),%eax

0x00000000004006e3 <+51>: jne 0x4006ea <p5+58>

0x00000000004006e5 <+53>: add $0x18,%rsp

0x00000000004006e9 <+57>: retq

0x00000000004006ea <+58>: xor %eax,%eax

0x00000000004006ec <+60>: callq 0x400690 <explode\_bomb>

End of assembler dump.

(gdb)

Answer the following, assuming you have reached a breakpoint at p5+16..

1. (1.5 points) The second parameter that is passed to the sscanf is a format string, which determines how sscanf interprets the string that is passed as its first parameter. What debugger command will display this format string?

* x/s $rsi

1. (1.5 points) p5 passes the address of a local variable as the third parameter to sscanf. Give the gdb command that reveals this address.

* x/x $rdx

1. (2 points) Assume that the debugger command in (a) displays %d as the format string. Given the results of the above, what is the input which will defuse p4 for the guest user ID? Explain your answer.

🡪 I have no idea. It looks like the line “cmp 0xc(%rsp), %eax” determines whether the bomb explodes or not seen as there is a jump if not equal command afterwards that jumps to the call <explode\_bomb>. Previously in the code whatever memory location that was stored in 0xc(%rsp) was moved into %rdx. But then %eax was moved into %rdx. All of this is confusing to me and I wish someone could explain it better.