

Andrew login ID: \_\_\_\_\_

Full Name: \_\_\_\_\_

Recitation Section: \_\_\_\_\_

## CS 15-213 / ECE 18-243, Spring 2010

### Exam 1

Version 1100101

Tuesday, March 2nd, 2010

#### Instructions:

- Make sure that your exam is not missing any sheets, then write your full name, Andrew login ID, and recitation section (A–J) on the front. Read all instructions and sign the statement below.
- Write your answers in the space provided for the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum of 100 points.
- The problems are of varying difficulty. We reserve the right to change the problems to be harder.
- You may not use any books or notes on this exam. Reference material is not allowed. No calculators or other electronic devices are allowed.
- Good luck!

I understand the CMU policy on cheating applies in full to this exam. \_\_\_\_\_

1- Multiple Choice (14):	
2- Peephole (16):	
3- Floating Point (14):	
4- Structs (14):	
5- Stacks (15):	
6- Buffer Overflow (17):	
7- Assembly (10):	
TOTAL (100):	

## Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

### Problem 1. (14 points):

1. Which of the following lines of C code performs the same operation as the assembly statement `lea 0xffffffff(%esi), %eax`.

(a) `*(esi-1) = eax`  
(b) `esi = eax + 0xffffffff`  
(c) `eax = esi - 1`  
(d) `eax = *(esi - 1)`

2. `test %eax, %eax`  
`jne 3d<function+0x3d>`

Which of the following values of `%eax` would cause the jump to be taken?

(a) 1  
(b) 0  
(c) Any value of `%eax`  
(d) No value of `%eax` would cause the jump to be taken.

3. Which of the following are legitimate advantages of x86\_64 over IA32? (Circle 0-3)

(a) x86\_64 is able to  
(b) x86\_64 is able to  
(c) x86\_64 is able to

4. T/F: Any sequence of IA32 instructions can be executed on an x86\_64 processor?

(a) True  
(b) False

5. What sequence of operations does the `leave` instruction execute?

(a) `mov %ebp, %esp`  
`pop %ebp`  
(b) `pop %ebp`  
`mov %ebp, %esp`  
(c) `pop %esp`  
`mov %ebp, %esp`  
(d) `push %ebp`  
`mov %esp, %ebp`

6. What is the difference between the `%rbx` and the `%ebx` register on an x86\_64 machine?
- (a) nothing, they are the same register
  - (b) `%ebx` refers to only the low order 32 bits of the `%rbx` register
  - (c) they are totally different registers
  - (d) `%ebx` refers to only the high order 32 bits of the `%rbx` register
7. On IA32 systems, where is the value of old `%ebp` saved in relation to the current value of `%ebp`?
- (a) there is no relation between where the current base pointer and old base pointer are saved.
  - (b) old `%ebp` is stored at `(%ebp - 4)`
  - (c) old `%ebp` is stored at `(%ebp + 4)`
  - (d) old `%ebp` is stored at `(%ebp)`

## Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`

## Problem 2. (16 points):

Consider the following assembly code:

```
08048334 <mystery>:
8048334: 55                push    %ebp
8048335: 89 e5             mov     %esp,%ebp
8048337: 83 ec 0c          sub     $0xc,%esp
804833a: 8b 45 08           mov     0x8(%ebp),%eax
804833d: c7 45 fc 00 00 00 movl    $0x0,0xffffffffc(%ebp)
8048344: 3b 45 fc           cmp     0xffffffffc(%ebp),%eax
8048347: 75 09             jne     8048352 <mystery+0x1e>
8048349: c7 45 f8 00 00 00 movl    $0x0,0xffffffff8(%ebp)
8048350: eb 12             jmp     8048364 <mystery+0x30>
8048352: 8b 45 08           mov     0x8(%ebp),%eax
8048355: 48               dec     %eax
8048356: 89 04 24           mov     %eax,(%esp)
8048359: e8 d6 ff ff ff    call    8048334 <mystery>
804835e: 03 45 08           add     0x8(%ebp),%eax
8048361: 89 45 f8           mov     %eax,0xffffffff8(%ebp)
8048364: 8b 45 f8           mov     0xffffffff8(%ebp),%eax
8048367: c9               leave
8048368: c3               ret
```

1. Fill in the blanks of the corresponding C function:

```
int mystery(int i)
{
    if (_____) return _____;

    return _____;
}
```

2. Peephole optimizations are a kind of optimization which looks at a small number of assembly instructions and tries to optimize those instructions. Care must be taken to not affect the behavior of the rest of the program. Write an optimized version of the assembly instructions at addresses `0x804833d` and `0x8048344`.
  
3. If we look at the addresses `0x8048361` and `0x8048364` it seems like we can eliminate both instructions or replace the instructions with nops. Explain why we can't implement this peephole optimization without affecting the behavior of the rest of the function.

## Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`

### Problem 3. (14 points):

Your friend, Harry Q. Bovik, encounters a function named `mystery` when running `gdb` on a 32-bit binary that was compiled on the fish machines. Use the `gdb` output below and the function prototype for `mystery` to complete this question.

```
int mystery(float arg1, float arg2, float arg3, float arg4);
```

```
Breakpoint 1, 0x08048366 in mystery ()
```

```
(gdb) x/20 $esp
```

```
0xffd3d1e0:    0xf7f3fff4      0xf7f3e204      0xffd3d208      0x080483cd
0xffd3d1f0:    0x41700000      0x3de00000      0x7f800010      0x00000001
0xffd3d200:    0x7f7fffff      0xffd3d220      0xffd3d278      0xf7e13e9c
0xffd3d210:    0xf7f5fca0      0x080483f0      0xffd3d278      0xf7e13e9c
0xffd3d220:    0x00000001      0xffd3d2a4      0xffd3d2ac      0xf7f60810
```

```
(gdb) print $ebp
```

```
$1 = (void *) 0xffd3d1e8
```

1. What is on the stack where `%ebp` is pointing (in hex)?

## Assignment Project Exam Help

2. What is the return address?  
<https://eduassistpro.github.io/>

## Add WeChat edu\_assist\_pro

Fill in the below table. Hexadecimal may be used in the address column. The value column may not contain any binary. Instead of calculating large powers of two you may use exponentials in the value column but your answer must fit within the table boundaries.

	address	value
arg1		
arg2		
arg3		
arg4		

**Problem 4. (14 points):**

Take the struct below compiled on Linux 32-bit:

```
struct my_struct {
    short b;
    int x;
    short s;
    long z;
    char c[5];
    long long a;
    char q;
}
```

1. Please lay out the struct in memory below (each cell is 1 byte). Please shade in boxes used for padding.

Assignment Project E

<https://eduassist.org>

Add WeChat edu

# Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro



Given the following gdb interaction (where ms is a struct my\_struct).

```
(gdb) x/40b &ms
0xfffffcde0: 0xbb 0x00 0x86 0x47 0xf9 0xd9 0x01 0x00
0xfffffcde8: 0x6d 0x3b 0xff 0xff 0xbe 0xba 0xef 0xbe
0xffffcdf0: 0x68 0x6c 0x70 0x6d 0x65 0x00 0x00 0x00
0xffffcdf8: 0x1e 0xab 0xdf 0x1e 0xff 0xe1 0xaf 0xde
0xffffce00: 0x21 0x00 0x00 0x00 0x00 0xf4 0x7f 0x86 0x47
```

2. Label the fields above and fill in the values below.

- ms.b = 0x\_\_\_\_\_
- ms.x = 0x\_\_\_\_\_
- ms.s = 0x\_\_\_\_\_
- ms.z = 0x\_\_\_\_\_
- ms.y = 0x\_\_\_\_\_
- ms.a = 0x\_\_\_\_\_
- ms.q = 0x\_\_\_\_\_

3. Define a struct with the same elements that has a total size of less than 30 b.

```
struct my_compressed_struct {
```

```
}
```

4. What is the size of my\_compressed\_struct that you wrote above?

### Problem 5. (15 points):

Below is the C code and assembly code for a simple function.

```
000000af <doSomething>:                                int doSomething(int a, int b, int c){
af:  push    %ebp                                       int d;
b0:  mov     %esp,%ebp                                  if (a == 0){ return 1;}
b2:  sub     $0xc,%esp                                  d = a/2;
b5:  mov     0x8(%ebp),%ecx                             c = doSomething(d,a,c);
b8:  mov     $0x1,%eax                                  return c;
bd:  test    %ecx,%ecx                                  }
bf:  je      de <doSomething+0x2f>
c1:  mov     %ecx,%edx
c3:  shr     $0x1f,%edx
c6:  lea     (%ecx,%edx,1),%edx
c9:  sar     %edx
cb:  mov     0x10(%ebp),%eax
ce:  mov     %eax,0x8(%esp)
d2:  mov     %ecx,0x4(%esp)
d6:  mov     %edx,(%esp)
d9:  call    da <doSomething+0x2b>
de:  leave
df:  ret
```

Please draw a detailed stack diagram that calls this function and continuing for 2 recursive calls of this function (that belong to this function). Please label everything you can.

ion  
es

<https://eduassistpro.github.io/>  
Add WeChat edu\_assist\_pro

[illegible]

## Problem 6. (17 points):

As a security engineer for a software company it is your job to perform attacks against your company's software and try to break it. One of your developers, Harry Q. Bovik, has written a password validator that he thinks is unbreakable! Below is the front-end to his system:

```
int main() {
    char buffer[20];

    printf("Enter your password >");
    scanf("%s", buffer);
    if (validate(buffer)) {
        getOnTheBoat();
        exit(0);
    }
    printf("Sorry, you do not have access :(\n");
    return 0;
}
```

**Step 0:** Briefly explain how you could attack this program with a buffer overflow. (25 words or less).

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Harry then mentions that you actually cannot perform that attack because where the stack is not-executable. This means that it is impossible to execute the typical attack you performed in buffer-lab now impossible.

Add WeChat edu\_assist\_pro

You can still do this though! You are going to perform a RETURN TO LIBC attack! This attack relies on pre-existing code in the program that will allow you to execute arbitrary instructions. There are a few important things you need to know about first:

The C function `system(char * command)` will execute the string `command` as if you had typed it into a shell prompt.

Using GDB you discover:

```
(gdb) print system
$1 = {<text variable, no debug info>} 0xf7e263a0 <system>
```

In every program executable, your environment variables are loaded at runtime. And part of your environment variables is your current SHELL:

```
(gdb) print (char *) 0xff89d957
$2 = 0xff89d957 "SHELL=/bin/bash"
```

Using this information, you can successfully launch a shell from Harry's program, proving that you can execute arbitrary code with his program's privilege level!

**Step 1:**

- What is the address of the `system()` function?
- What is the address of the string `"/bin/bash"`?

**Step 2:** Design your exploit string (keep in mind where arguments go for IA32 ). We're looking for an drawing of what you can pass as input to this program causing it to launch a shell. Don't worry about exact sizes/lengths.

## Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

**Step 3:** Explain how your exploit string will allow you to execute a shell. Combined with your answer to Step 2 should be enough to prove Harry wrong. (This will be graded independently of your Step 2).

### Problem 7. (10 points):

Use the x86\_64 assembly to fill in the C function below

```
0x0000000000400498 <mystery+0>:    push    %r13
0x000000000040049a <mystery+2>:    push    %r12
0x000000000040049c <mystery+4>:    push    %rbp
0x000000000040049d <mystery+5>:    push    %rbx
0x000000000040049e <mystery+6>:    sub     $0x8,%rsp
0x00000000004004a2 <mystery+10>:   mov     %rdi,%r13
0x00000000004004a5 <mystery+13>:   mov     %edx,%r12d
0x00000000004004a8 <mystery+16>:   test    %edx,%edx
0x00000000004004aa <mystery+18>:   jle     0x4004c7 <mystery+47>
0x00000000004004ac <mystery+20>:   mov     %rsi,%rbx
0x00000000004004af <mystery+23>:   mov     $0x0,%ebp
0x00000000004004b4 <mystery+28>:   mov     (%rbx),%edi
0x00000000004004b6 <mystery+30>:   callq   *%r13
0x00000000004004b9 <mystery+33>:   mov     %eax, (%rbx)
0x00000000004004bb <mystery+35>:   add     $0x1,%ebp
0x00000000004004be <mystery+38>:   add     $0x4,%rbx
0x00000000004004c3 <mystery+42>:   cmp     %r12d,%ebp
0x00000000004004c5 <mystery+45>:   jne     0x4004b4 <mystery+28>
0x00000000004004cd <mystery+53>:   pop     %r12
0x00000000004004cf <mystery+55>:   pop     %r13
0x00000000004004d1 <mystery+57>:   retq
```

```
void mystery(int (*funcP)(int), int a[], int n) {

}
```