Q1 Course Policy

3 Points

To support remote learning and reduce your workload this semester, homework assignments this semester have instant feedback enabled. When you click "Save Answer," if the answer is correct, you will see an explanation.

You can resubmit as many times as you want until the due date. After the due date, to avoid being marked late, do not submit again.

Relevant lecture: Tuesday, January 18: Introduction (**course policies**, **slides**, **recording**)

Q1.1 Slip Days

1 Point

How many project/lab slip days do you get? Answer as a number (e.g. `5`).

0

How many homework slip days do you get? Answer as a number (e.g. `5`).

0

Q1.2 Collaboration

1 Point

You're working on a course project. Your code isn't working, and you can't figure out why not. Is it OK to show another student (who is not your project partner) your draft code and ask them if they have any idea why your code is broken or any suggestions for how to debug it?

- ○ Yes
- ◉ No
- ○ As long as they don't tell you the answer

Q1.3 Readings
1 Point

Which of the following readings are in scope for exams? Select all that apply.

- ☑ Live or pre-recorded lectures
- ☐ Staff-created course textbook
- ☑ Discussion worksheets
- ☐ None of the above

Q2 Course Accounts
2 Points

Follow the steps in this question to set up the accounts and install the software you need for this course.

Q2.1 Piazza
1 Point

If you have not already been added to Piazza, please email cs161-staff@berkeley.edu. We recommend using the same name on Piazza and Gradescope.

Once you have an account, find the Homework 1 post and enter the password posted there as the answer to this question. In accordance with the class policies on doing work individually, do not share the password with anyone.

correcthorsebatterystaple

Q2.2 Instructional Account

1 Point

Create an EECS instructional class account if you have not already. To do so, visit **the EECS web account page**, click “Login using your Berkeley CalNet ID,” then find the cs161 row and click “Get a new account.” Be sure to take note of the account login and password.

Please ssh into your account by running `ssh cs161-xxx@hiveyy.cs.berkeley.edu`, replacing `xxx` with the last three letters of your account and `yy` with a number between 1 and 30.

Once you've logged into your account, run `more /share/b/pub/disk.quotas`. You don't need to read the document; just enter the date you see on the fourth line of the document. Answer in the format `Jan 18, 2021`.

If you are having trouble with getting an instructional account, please make a private post on Piazza.

Feb 24, 2017

Q3 Security Principles

5 Points

Relevant lecture: Thursday, January 20: Security Principles (**textbook**, **slides**, **recording**)

For each of the following paragraphs, select the security principle that **best** applies to the situation described.

Bob one day decides to set up his own free-to-use pet-photo website.

Q3.1

1 Point

He sets up all the infrastructure himself, but worries about forgetting the admin password. Bob hides his login credentials in a long HTML comment on the login page.

- Bob forgets that security is economics
- Bob uses fail-unsafe defaults
- Bob violates least privilege
- Bob violates the separation of responsibility
- (selected) Bob relies on security through obscurity
- Bob violates complete mediation
- Bob fails to consider human factors
- Bob fails to design security in from the start
- Bob fails to employ defense-in-depth
- Bob violates detect if you can't prevent
- Bob does not consider his threat model

Q3.2

1 Point

Bob hires Mallory's Do-No-Evil design firm to design the website front-end. He gives them an account with access to his front-end and back-end codebase, and databases of user information as well.

- Bob forgets that security is economics
- Bob uses fail-unsafe defaults
- (selected) Bob violates least privilege
- Bob violates the separation of responsibility
- Bob relies on security through obscurity
- Bob violates complete mediation
- Bob fails to consider human factors
- Bob fails to design security in from the start
- Bob fails to employ defense-in-depth
- Bob violates detect if you can't prevent
- Bob does not consider his threat model

Q3.3

1 Point

Finally, Bob wants to enforce password security. Bob requires every user to use a "super-secure" password: the password cannot contain any English word, cannot contain any birthday, and must have many special characters (e.g., $ \%).

The user needs to type in this password every 5 minutes. Bob disables the clipboard on the password field. Therefore, the user must manually enter the password.

- ○ Bob forgets that security is economics
- ○ Bob uses fail-unsafe defaults
- ○ Bob violates least privilege
- ○ Bob violates the separation of responsibility
- ○ Bob relies on security through obscurity
- ○ Bob violates complete mediation
- ◉ Bob fails to consider human factors
- ○ Bob fails to design security in from the start
- ○ Bob fails to employ defense-in-depth
- ○ Bob violates detect if you can't prevent
- ○ Bob does not consider his threat model

Q3.4

1 Point

Bob one day wakes up to his website being featured on a well-known news site after a data leak. He panics, and spends millions of dollars hiring a security consultant to find all of his vulnerabilities.

- [x] Bob forgets that security is economics
- [ ] Bob uses fail-unsafe defaults
- [ ] Bob violates least privilege
- [ ] Bob violates the separation of responsibility
- [ ] Bob relies on security through obscurity
- [ ] Bob violates complete mediation
- [ ] Bob fails to consider human factors
- [ ] Bob fails to design security in from the start
- [ ] Bob fails to employ defense-in-depth
- [ ] Bob violates detect if you can't prevent
- [ ] Bob does not consider his threat model

Q3.5

1 Point

With all the vulnerabilities identified, Bob starts trying to fix his poor site. Unfortunately, much of the original code was written in a late-night, coffee-fueled frenzy, and Bob finds that he can't fix any aspect of the website without breaking the rest in its entirety.

Bob announced the closure of his site and goes into hiding.

- [ ] Bob forgets that security is economics
- [ ] Bob uses fail-unsafe defaults
- [ ] Bob violates least privilege
- [ ] Bob violates the separation of responsibility
- [ ] Bob relies on security through obscurity
- [ ] Bob violates complete mediation
- [ ] Bob fails to consider human factors
- [x] Bob fails to design security in from the start
- [ ] Bob fails to employ defense-in-depth
- [ ] Bob violates detect if you can't prevent
- [ ] Bob does not consider his threat model

Q4 C memory review

3 Points

Relevant lecture: Thursday, January 20: Security Principles and x86 Assembly(**textbook**, **slides**, **recording**)

The next three questions introduce some CS61C-related concepts you need for Project 1. **CS61C review resources** are available if you need a quick review.

Consider the three code snippets below. Which variable is located at a higher address in memory?

Q4.1

1 Point

```
int x;
char y[4];
int z;

int main() {
    return 0;
}
```

- ○ `x`
- ○ `y`
- ◉ `z`

Q4.2

1 Point

```
int main() {
    int x;
    char y[4];
    int z;

    return 0;
}
```

- (•) `x`
- ( ) `y`
- ( ) `z`

Q4.3
1 Point

```
struct s {
    int x;
    char y[4];
    int z;
};

int main() {
    struct s foo;

    return 0;
}
```

- ( ) `x`
- ( ) `y`
- (•) `z`

Q5 C stack layout
8 Points

Relevant lecture: Thursday, January 20: Security Principles and x86 Assembly(**textbook**, **slides**, **recording**)

A note on terminology: Suppose a function `foo` calls a function `bar`.

- "The **ebp** of `bar`" refers to the value in the ebp register while `bar` is executing.
- Similarly, "the **eip** of `bar`" refers to the value in the eip register while `bar` is executing.
- "The **sfp** of `bar`" refers to the ebp of `foo` (the caller) that is saved on the stack while calling `bar` (the callee).

- Similarly, "the **rip** of `bar`" refers to the eip of `foo` (the caller) that is saved on the stack while calling `bar` (the callee).

Please draw the stack at the indicated point in code execution. We recommend drawing out the stack on paper, and then filling out the multiple-choice options once you have the diagram drawn.

```
int main() {
     f();
}

void f() {
    char s[] = "abc";
    g(s);
}

void g(char *s) {
    /*  DRAW THE STACK HERE */
    /* ...more code here... */
}
```

Q5.1

1 Point

Starting at the **highest** address and going down, match one option per subquestion. Each option appears exactly once.

(Your answer to this part should be the option with the highest address.)

- ○ `sfp` of `main`
- ○ `sfp` of `f`
- ○ `sfp` of `g`
- ◉ `rip` of `main`
- ○ `rip` of `f`
- ○ `rip` of `g`
- ○ `"abc\0"`
- ○ The address of `"abc\0"`

Q5.2

1 Point

- [x] `sfp` of `main`
- [ ] `sfp` of `f`
- [ ] `sfp` of `g`
- [ ] `rip` of `main`
- [ ] `rip` of `f`
- [ ] `rip` of `g`
- [ ] `"abc\0"`
- [ ] The address of `"abc\0"`

Q5.3
1 Point

- [ ] `sfp` of `main`
- [ ] `sfp` of `f`
- [ ] `sfp` of `g`
- [ ] `rip` of `main`
- [x] `rip` of `f`
- [ ] `rip` of `g`
- [ ] `"abc\0"`
- [ ] The address of `"abc\0"`

Q5.4
1 Point

- ○ `sfp` of `main`
- ◉ `sfp` of `f`
- ○ `sfp` of `g`
- ○ `rip` of `main`
- ○ `rip` of `f`
- ○ `rip` of `g`
- ○ `"abc\0"`
- ○ The address of `"abc\0"`

Q5.5
1 Point

- ○ `sfp` of `main`
- ○ `sfp` of `f`
- ○ `sfp` of `g`
- ○ `rip` of `main`
- ○ `rip` of `f`
- ○ `rip` of `g`
- ◉ `"abc\0"`
- ○ The address of `"abc\0"`

Q5.6
1 Point

- `sfp` of `main`
- `sfp` of `f`
- `sfp` of `g`
- `rip` of `main`
- `rip` of `f`
- `rip` of `g`
- `"abc\0"`
- The address of `"abc\0"`

Q5.7
1 Point

- `sfp` of `main`
- `sfp` of `f`
- `sfp` of `g`
- `rip` of `main`
- `rip` of `f`
- `rip` of `g`
- `"abc\0"`
- The address of `"abc\0"`

Q5.8
1 Point

(Your answer to this part should be the option with the lowest address.)

- ○ `sfp` of `main`
- ○ `sfp` of `f`
- ◉ `sfp` of `g`
- ○ `rip` of `main`
- ○ `rip` of `f`
- ○ `rip` of `g`
- ○ `"abc\0"`
- ○ The address of `"abc\0"`

Q6 More C stack layout

4 Points

Relevant lecture: Thursday, January 20: x86 Assembly and Call Stack (**textbook**, **slides**, **recording**)

Consider the following C code:

```
void foo(int arg) {
    long long longVar; //takes up 8 bytes
    int intVar;
    float floatVar;
    bar(arg);
}

void bar(int arg) {
    char ramble[60] = "I wonder if a foobar tastes better than a c
    //breakpoint
    if (arg) {
        printf(ramble);
    }
}

int main() {
    int hungry = 1;
    foo(hungry);
    return 0;
}
```

In this class, unless otherwise stated, always assume that we're running C code on a 32-bit, little-endian x86 architecture.

For this question, additionally assume that there are no stack canaries, no exception handlers, no callee saved registers, and no compiler optimizations. (Don't worry if you don't know what these are yet.)

Exception handlers, callee saved registers, and compiler optimizations often appear in practice, as you will see in Project 1. This means gdb cannot be used to solve this question - instead, consider drawing the stack diagram, like in the previous question.

In this class, the location of a variable is the lowest address associated with it, i.e., the address of the start of the memory region where it is stored. In a hypothetical stack frame with these three variables,

```
   |__________|___HIGH
X  |_4_bytes__|
Y  |_8_bytes__|
Z  |_12_bytes_|___LOW
```

We define `X` to be 8 bytes above `Y` and 20 bytes above `Z`.

Answer the questions below using the stack layout of the code, assuming it started by executing `main` and is now at the `breakpoint` in `bar`.

Q6.1

1 Point

The `rip` of `main` is located how many bytes above local variable `hungry`?

8

Q6.2

1 Point

The `sfp` of `foo` is located how many bytes above `longVar`?

8

Q6.3
1 Point

The `sfp` of `foo` is located how many bytes above the `rip` of `bar`?

24

Q6.4
1 Point

The `rip` of `main` is located how many bytes above `ramble`?

108

Q7 gdb
6 Points

Project 1 requires you to be familiar with the gdb C debugger. For this question, we recommend you set up the Project 1 VM, complete the customization step, and log into the first question `remus`. (See the setup pages of the **project spec** for setup instructions.)

Once you've finished setting up the Project 1 VM, start the debugger by running `./debug-exploit`.

Note: because your VM will be customized to have different addresses, the numbers in this question might not be identical to the numbers on your VM. However, we still encourage you to play around with gdb to prepare for Project 1.

Q7.1
1 Point

You want to see investigate the program state at line 5. What sequence of commands should you run in gdb?

- (●) `b 5`, then `r`
- ( ) `r`, then `b 5`
- ( ) `r 5`, then `b`
- ( ) `b`, then `r 5`

Q7.2

1 Point

Run `i f` (short for `info frame`). The main pieces of information here are the last two lines:

```
Saved registers:
  ebp at 0xbffffe08, eip at 0xbffffe0c
```

What do these values represent?

Hint: these two values are 4 bytes apart. Which of these options do you expect to be 4 bytes apart?

- ( ) The addresses of the `ebp` and `eip` registers.
- ( ) The values in the `ebp` and `eip` registers.
- ( ) The values of `sfp` and `rip` on the stack.
- (●) The addresses of `sfp` and `rip` on the stack.

Q7.3

1 Point

If you run `x buf`, you should see output similar to:

```
(gdb) x buf
0xbffffdf8:  0xbffffeac
```

Hint: `x/nxw addr` prints out `n` hex words of memory starting at `addr`.

What does the number on the left, `0xbffffdf8` represent?

- [x] The address of `buf`
- [ ] The first 4 bytes of the value in `buf`
- [ ] The value at the address stored in `buf`

What does the number on the right, `0xbffffeac` represent?

- [ ] The address of `buf`
- [x] The first 4 bytes of the value in `buf`
- [ ] The value at the address stored in `buf`

Q7.4

1 Point

Again, suppose you see this output:

```
(gdb) x buf
0xbffffdf8:  0xbffffeac
```

Hint 1: remember that x86 is **little-endian**.
Hint 2: `x/nxb addr` prints out `n` bytes of memory starting at `addr`.

What is the value of the byte located at address `0xbffffdf8`?

- [ ] 0xbf
- [ ] 0xff
- [ ] 0xfe
- [x] 0xac
- [ ] Not enough information

What is the value of the byte located at address `0xbffffdfa`?

- [ ] 0xbf
- [x] 0xff
- [ ] 0xfe
- [ ] 0xac
- [ ] Not enough information

Q7.5

1 Point

If you run `x $ebp`, you should see output similar to:

```
(gdb) x $ebp
0xbffffe08:  0xbffffe18
```

What does the number on the left, `0xbffffe08` represent?

Hint: Have you seen this number `0xbffffe08` before?

- ○ The address of `ebp`
- ◉ The value in `ebp`
- ○ The value at the address stored in `ebp`

What does the number on the right, `0xbffffe18` represent?

- ○ The address of `ebp`
- ○ The value in `ebp`
- ◉ The value at the address stored in `ebp`

Q7.6

1 Point

Next, run `x/4xw buf`. You should see output similar to:

```
(gdb) x/4xw buf
0xbffffdf8:  0xbffffeac  0xb7ffc165  0x00000000  0x00000000
```

Let's label each word of the output as follows:

```
(gdb) x/4xw buf
(A)0xbffffdf8:  (B)0xbffffeac  (C)0xb7ffc165  (D)0x00000000  (E)0x
```

Which two words correspond to the 8 bytes stored in `buf`?

- [ ] A
- [x] B
- [x] C
- [ ] D
- [ ] E

Q8 A simple buffer overflow

2 Points

Relevant lecture: Tuesday, January 25: Memory Safety Vulnerabilities (**textbook**, **slides (TBD)**, **recording (TBD)**)

Consider the following vulnerable C code.

```
void process_query(char *input) {
    char str[4];
    strncpy(str, input, strlen(input)+1);
    /* . . . */
}
```

Suppose the attacker has placed machine code at memory address `0xdeadbeef`. What input should you give to `process_query` so that the code at `0xdeadbeef` is executed?

Q8.1

1 Point

First, write ___ bytes of garbage. Your answer should be a number.

Hint: If you are having trouble, consider drawing the stack diagram.

8

Q8.2

1 Point

Next, write these four hex bytes. Your answer should be four bytes in Python/Project 1 format (e.g. `\xaa\xbb\xcc\xdd`).

Hint: If you are having trouble, remember that x86 is **little-endian**.

\xef\xbe\xad\xde

Q9 Feedback

0 Points

Optionally, feel free to include feedback. What's something we could do to make the class better? Or, what did you find most difficult or confusing from lectures or the rest of class, and what would you like to see explained better? If you have feedback, submit your comments here.

Your name will not be connected to any feedback you provide, and anything you submit here will not affect your grade.

Homework 1

● GRADED

STUDENT
Ko Tsun Leung

TOTAL POINTS
33 / 33 pts

QUESTION 1

Course Policy — **3** / 3 pts

1.1 Slip Days — **1** / 1 pt

1.2	Collaboration	**1** / 1 pt
1.3	Readings	**1** / 1 pt

QUESTION 2

Course Accounts **2** / 2 pts

2.1	Piazza	**1** / 1 pt
2.2	Instructional Account	**1** / 1 pt

QUESTION 3

Security Principles **5** / 5 pts

3.1	(no title)	**1** / 1 pt
3.2	(no title)	**1** / 1 pt
3.3	(no title)	**1** / 1 pt
3.4	(no title)	**1** / 1 pt
3.5	(no title)	**1** / 1 pt

QUESTION 4

C memory review **3** / 3 pts

4.1	(no title)	**1** / 1 pt
4.2	(no title)	**1** / 1 pt
4.3	(no title)	**1** / 1 pt

QUESTION 5

C stack layout **8** / 8 pts

5.1	(no title)	**1** / 1 pt
5.2	(no title)	**1** / 1 pt
5.3	(no title)	**1** / 1 pt
5.4	(no title)	**1** / 1 pt
5.5	(no title)	**1** / 1 pt
5.6	(no title)	**1** / 1 pt
5.7	(no title)	**1** / 1 pt
5.8	(no title)	**1** / 1 pt

QUESTION 6

More C stack layout **4** / 4 pts

6.1	(no title)	**1** / 1 pt
6.2	(no title)	**1** / 1 pt
6.3	(no title)	**1** / 1 pt

6.4 (no title) **1** / 1 pt

QUESTION 7

gdb **6** / 6 pts

7.1 (no title) **1** / 1 pt

7.2 (no title) **1** / 1 pt

7.3 (no title) **1** / 1 pt

7.4 (no title) **1** / 1 pt

7.5 (no title) **1** / 1 pt

7.6 (no title) **1** / 1 pt

QUESTION 8

A simple buffer overflow **2** / 2 pts

8.1 (no title) **1** / 1 pt

8.2 (no title) **1** / 1 pt

QUESTION 9

Feedback **0** / 0 pts