34/35 Questions Answered
Saved at 12:04 PM

Homework 1

Q1 Course Policy
4 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.

Open the course policies from the class website. Read them carefully and answer the following questions:

Q1.1 Slip Days
1 Point

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

6

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

0

EXPLANATION

Note that you get one homework **drop**, and 6 **slip days** for projects and labs.

No late homeworks are accepted. (You should only use the late due date for homeworks if you have an extension.)

✔ **Correct**

Save Answer Last saved on **Jan 19 at 6:20 PM**

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

EXPLANATION

No, it is never acceptable to show another student your code, even if it is just a draft.

✔ **Correct**

Save Answer Last saved on **Jan 19 at 6:21 PM**

Q1.3 Readings

1 Point

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

- [x] Staff-created course notes
- [ ] *Introduction to Computer Security* by Goodrich & Tamassia
- [ ] *The Craft of System Security* by Smith & Marchesini
- [x] Discussion worksheets
- [ ] None of the above

EXPLANATION

If you'd like to use the optional textbooks, you can see suggested readings from them by clicking "Show/hide optional readings" on the course homepage.

✔ **Correct**

Save Answer Last saved on **Jan 19 at 6:23 PM**

Q1.4 Optional Labs

1 Point

Which of the following scores can be used to replace your lab grade? Select all that apply.

- [ ] Exam scores
- [x] Project scores
- [x] Homework scores

EXPLANATION

Your final lab score is the maximum of your overall lab score, overall project score, and overall homework score. If you don't do the optional labs, then your final lab score is the higher of your project and homework scores.

Note that if you try the labs and your project or homework score is higher than your lab scores, this policy still applies (you get the maximum of the 3 scores). The labs are purely for your learning/enjoyment, so don't stress too much about them!

✔ Correct

Save Answer Last saved on **Jan 19 at 6:25 PM**

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 add yourself by following this signup link. 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

EXPLANATION

correcthorsebatterystaple

✔ Correct

Save Answer Last saved on **Jan 19 at 6:26 PM**

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

EXPLANATION

Feb 24, 2017

✔ **Correct**

Save Answer Last saved on **Jan 19 at 6:49 PM**

Q3 Security Principles

5 Points

Relevant lecture: Thursday, January 21, Security Principles (slides, recording, review videos, notes)

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
- ◉ 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

EXPLANATION

Bob should assume that the attacker knows the entire system and can see any HTML comments.

✔ **Correct**

Save Answer Last saved on **Jan 22 at 10:32 AM**

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
- ◉ 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

EXPLANATION

Mallory's design firm is only doing front-end design work, so they do not need access to the back-end codebase or the databases of user information.

✔ **Correct**

Save Answer Last saved on **Jan 22 at 12:03 PM**

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

EXPLANATION

This implementation is very user-unfriendly. For example, users might write the complicated passwords down to avoid forgetting them.

✔ **Correct**

Save Answer Last saved on **Jan 22 at 2:25 PM**

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.

- ◉ 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

EXPLANATION

Bob's pet photo website is probably not worth millions of dollars. Bob is using a million-dollar lock to secure a thousand-dollar website.

✔ Correct

Save Answer Last saved on **Jan 22 at 2:32 PM**

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
- ◉ Bob fails to design security in from the start

EXPLANATION

The late-night code writing, many vulnerabilities, and difficulty in fixing those vulnerabilites suggests Bob did not consider security when writing the original code.

✔ Correct

Save Answer Last saved on **Jan 22 at 2:33 PM**

Q4 C memory review

3 Points

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`

EXPLANATION

The variables are declared in the **static** portion of memory, which is filled in from the bottom up. This means `x` has the lowest address, `y` has a higher address, and `z` has the highest address.

✔ Correct

Save Answer Last saved on **Jan 21 at 12:40 AM**

Q4.2

1 Point

```
int main() {
    int x;
```

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

    return 0;
}
```

- (•) `x`
- ( ) `y`
- ( ) `z`

EXPLANATION

The variables are declared on the **stack** in memory, which starts at higher addresses and grows downwards. This means `x` has the highest address, `y` has a lower address, and `z` has the lowest address.

✔ Correct

Save Answer Last saved on **Jan 21 at 1:06 AM**

Q4.3

1 Point

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

int main() {
    struct s foo;

    return 0;
}
```

- ( ) `x`
- ( ) `y`
- (•) `z`

EXPLANATION

A **struct** is always treated as a group of variables, no matter what part of memory it is in. Even though the struct here is on the **stack**, which grows downwards, within the struct, `x` has the lowest address, `y` has a higher address, and `z` has the highest address.

If another variable was declared after the line `struct s foo;`, that variable would be located below `x` on the stack.

✔ Correct

Save Answer Last saved on **Jan 21 at 1:06 AM**

Q5 C stack layout

8 Points

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"`

EXPLANATION

`rip` of `main`

✔ Correct

Save Answer Last saved on **Jan 21 at 6:13 PM**

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"`

EXPLANATION

`sfp` of `main`

✔ Correct

Save Answer Last saved on **Jan 21 at 6:14 PM**

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"`

EXPLANATION

`rip` of `f`

✔ Correct

Save Answer Last saved on **Jan 21 at 6:14 PM**

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"`

EXPLANATION

`sfp` of `f`

✔ Correct

Save Answer Last saved on **Jan 21 at 6:14 PM**

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"`

EXPLANATION

This is the local variable in `f`.

✔ Correct

Save Answer Last saved on **Jan 21 at 6:14 PM**

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"`

EXPLANATION

This is the pointer `*s` passed as an argument to `g`.

✔ Correct

Save Answer Last saved on **Jan 21 at 6:14 PM**

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"`

EXPLANATION

`rip` of `g`

✔ Correct

Save Answer Last saved on **Jan 21 at 6:15 PM**

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"`

EXPLANATION

`sfp` of `g`

✔ Correct

Save Answer Last saved on **Jan 21 at 6:15 PM**

Q6 More C stack layout

4 Points

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 chocolate
    //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

EXPLANATION

```
[4]  rip of main          *   (points wherever main returns to)
[4]  sfp of main              (points to wherever ebp was before main)
[4]  int hungry           *   (local var of main)
```

(rest of the stack omitted)

✔ Correct

Save Answer Last saved on **Jan 21 at 6:17 PM**

Q6.2

1 Point

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

8

EXPLANATION

```
[4]  rip of main              (points wherever main returns to)
[4]  sfp of main              (points to wherever ebp was before main)
[4]  int hungry               (local var of main)
[4]  int arg                  (argument passed from main to foo)
[4]  rip of foo               (points to eip of main)
[4]  sfp of foo           *   (points to ebp of main)
[8]  long long longVar    *   (local var of foo)
```

(rest of the stack omitted)

✔ Correct

Save Answer Last saved on **Jan 21 at 6:18 PM**

Q6.3

1 Point

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

24

EXPLANATION

```
[4]  rip of main                (points wherever main returns to)
[4]  sfp of main                (points to wherever ebp was before main)
[4]  int hungry                 (local var of main)
[4]  int arg                    (argument passed from main to foo)
[4]  rip of foo                 (points to eip of main)
[4]  sfp of foo           *     (points to ebp of main)
[8]  long long longVar          (local var of foo)
[4]  int intVar                 (local var of foo)
[4]  float floatVar             (local var of foo)
[4]  int arg                    (argument passed from foo to bar)
[4]  rip of bar           *     (points to eip of foo)
```

(rest of the stack omitted)

✔ Correct

Save Answer Last saved on **Jan 21 at 6:19 PM**

Q6.4

1 Point

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

108

EXPLANATION

Here is the entire stack:

```
[4]  rip of main          *   (points wherever main returns to)
[4]  sfp of main              (points to wherever ebp was before main)
[4]  int hungry               (local var of main)
[4]  int arg                  (argument passed from main to foo)
[4]  rip of foo               (points to eip of main)
[4]  sfp of foo               (points to ebp of main)
[8]  long long longVar        (local var of foo)
[4]  int intVar               (local var of foo)
[4]  float floatVar           (local var of foo)
[4]  int arg                  (argument passed from foo to bar)
[4]  rip of bar               (points to eip of foo)
[4]  sfp of bar               (points to ebp of foo)
[60] ramble               *   (local var of bar)
```

✔ Correct

Save Answer Last saved on **Jan 21 at 6:21 PM**

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 first 3 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`

EXPLANATION

`b 5` (short for `break 5`) sets a breakpoint at line 5. `r` (short for `run`) starts program execution.

✔ Correct

Save Answer Last saved on **Jan 23 at 11:25 AM**

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.

EXPLANATION

Option 1: The `ebp` and `eip` registers are located on the CPU, which doesn't have any addresses at all.

Option 2: The `ebp` register has an address of a location on the stack. The `eip` register has an address of a location in the code section. These values are much more than 4 bytes apart.

Option 3: `sfp` points to the previous `ebp` (on the stack), and `rip` points to the previous `eip` (in the code section), so these values are probably not 4 bytes apart.

Option 4: In a stack diagram, `sfp` and `rip` are located next to each other (4 bytes apart).

✔ **Correct**

Save Answer Last saved on **Jan 23 at 11:26 AM**

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?

- (●) 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`
- (●) The first 4 bytes of the value in `buf`
- ( ) The value at the address stored in `buf`

EXPLANATION

Running `x var` gives you the address of `var` on the left, and the value of `var` on the right.

In this case, `buf` may not contain an address at all, so option 3 doesn't really make sense.

✔ **Correct**

Save Answer Last saved on **Jan 23 at 11:28 AM**

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
- ◉ 0xac
- ○ Not enough information

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

- ○ 0xbf
- ◉ 0xff
- ○ 0xfe
- ○ 0xac
- ○ Not enough information

EXPLANATION

x86 is **little-endian**, which means within each word, the least significant byte is stored first.

The byte at `0xbffffdf8` is `0xac`.
The byte at `0xbffffdf9` is `0xfe`.
The byte at `0xbffffdfa` is `0xff`.
The byte at `0xbffffdfb` is `0xbf`.

Note: This means if you want to write a 4-byte address like `0xdeadbeef` onto the stack, you should enter it as `\xef\xbe\xad\xde` in your Project 1 code.

✔ **Correct**

Save Answer Last saved on **Jan 23 at 11:32 AM**

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`

EXPLANATION

The `ebp` register is located on the CPU, which doesn't have any addresses at all. So option 1 doesn't really make sense.

One way to reason this out: Recall that `ebp` always points to `sfp`. In Q7.2 you determined that `i f` gives you the address of `sfp`. Notice that the same value appears on the left side here, so it must be the value of `ebp`.

Then, you can run `x 0xbffffe08` to confirm that the address on the left side contains the value on the right side. (This is consistent with how `x` always displays addresses and values.)

✔ **Correct**

Save Answer Last saved on **Jan 23 at 11:38 AM**

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)0x00000000
```

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

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

EXPLANATION

`buf` is 8 bytes = 2 words, so we know two options should be checked. (A) is an address. Since we ran `x/4xw buf`, the memory will start printing exactly at `buf`. Thus `buf` contains the first two words printed from memory, which are (B) and (C).

✔ **Correct**

Save Answer Last saved on **Jan 23 at 11:51 AM**

Q8 A simple buffer overflow

2 Points

Relevant lecture: Tuesday, January 26, Buffer Overflows (slides TBA, recording TBA, review videos, notes)

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

EXPLANATION

The stack diagram looks like:

`rip` of `process_query`
`sfp` of `process_query`
`str[4]`

The `strncpy` line writes your input into `str`, but it lets you write past the end of `str` as far as you want. From the diagram, we need to write 8 bytes of garbage: 4 bytes to overwrite the `str` array, and 4 bytes to overwrite the `sfp`.

✔ Correct

Save Answer Last saved on **Jan 23 at 11:59 AM**

Q8.2

1 Point

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

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

\xef\xbe\xad\xde

EXPLANATION

To start executing code at `0xdeadbeef`, we want to overwrite the `rip` with `0xdeadbeef`. However, remember that x86 is **little-endian**, so we have to input the bytes backwards so that the program will interpret the resulting word as `0xdeadbeef`.

✔ Correct

Save Answer Last saved on **Jan 23 at 12:04 PM**

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.

Enter your answer here

Save Answer

Save All Answers Submit & View Submission >