Popa & Weaver
Spring 2019

CS 161 Computer Security

Homework 1

Due: Tuesday, 5 February 2019, at 11:59pm

Instructions. This homework is due on **Tuesday, 5 February 2019, at 11:59pm**. No late homeworks will be accepted unless you have prior accomodations from us. This assignment must be done on your own.

Create an EECS instructional class account if you have not already. To do so, visit https://inst.eecs.berkeley.edu/webacct/, 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, and log in to your instructional account.

Make sure you have a Gradescope account and are joined in this course. The homework *must* be submitted electronically via Gradescope (not by any other method). Your answer for each question, when submitted on Gradescope, should be a single file with each question's answer on a separate page.

Problem 1 ***Policy*** **(10 points)**

The aim of this exercise is to ensure that you read the course policies, as well as to make sure that you are registered in the class and have a working EECS instructional class account.

Open the course website http://inst.eecs.berkeley.edu/~cs161/sp19/.

Read the course policies and answer the following question:

How many project "slip days" do you get?

0

Problem 2 ***Collaboration*** **(10 points)**

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?

Select if yes ☐

Problem 3 ***Security Principles*** **(20 points)**

For each of the following paragraphs, there is exactly one security principle that best applies to the situation described. Select the best **four options** from below after reading the following scenario:

Getting on the cryptocurrency hype, one day Bob decides to set up his own exchange. He sets up all the infrastructure, but worries about forgetting the password, so Bob hides his login credentials in an HTML comment on the login page.

Eventually, Bob manages to gather a large user-base and realizes his site looks like a back-end developer trying to learn CSS, so he contracts out front-end work to Mallory's Do-No-Evil design firm (for an incredible price too!). He gives them an account with access to his front-end and back-end codebase, and databases of user information as well.

Finally, Bob wants to enforce password security. Bob requires every user to use a "super-secure" password; that is, 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; in this way, the user must manually enter the password, nothing else.

Unfortunately for him, one day he wakes up to his website being featured on a well-known news site after a data leak. Pressured by an internet mob, he hires a contractor to find all the issues with his site. However, fixing the website ended up being a different story, as much of the code was written (uncommented) in a late-night coffee-fuled frenzy, and Bob finds that he can't change any aspect of the website without breaking it in its entirety. In a panic, Bob announced the closure of his site and goes into hiding.

1. Does Bob violate **security is economics**? Select if yes ☐
2. Does Bob violate **least privilege**? Selected! ☑
3. Does Bob violate **fail-safe defaults**? Select if yes ☐
4. Does Bob violate **separation of responsibility**? Select if yes ☐
5. Does Bob violate **don't rely on security by obscurity**? Selected! ☑
6. Does Bob violate **consider human factors**? Selected! ☑
7. Does Bob violate **complete mediation**? Select if yes ☐
8. Does Bob violate **detect if you can't protect**? Select if yes ☐
9. Does Bob violate **design security in from the start**? Selected! ☑

Problem 4 ***Vulnerable Code*** **(40 points)**

Consider the following C code:

```
1  void greet(char *arg)
2  {
3    char buffer[12];
4    printf(``I am the Senate. What is your name?\n");
5    scanf(``%s", buffer);
6    printf(``It's treason then, %s\n", buffer);
7  }
8
9  int main(int argc, char *argv[])
10 {
11   char beg[3] = 'Obi';
12   char end[11] = 'Wan Kenobi?';
13   strncat(beg, end, 5);
14   greet(argv[1]);
15   return 0;
16 }
```

1. What is the line number that has a memory vulnerability?

   5

2. What is this vulnerability called?

   buffer overflow attack

3. Just before the program executes the line in part 1, the registers are:

   `%esp: 0xBFFFF820` `%ebp: 0xBFFFF848`

   Given this information, describe in detail how an attacker would take advantage of the vulnerability. Also make sure to include the address that the attacker needs to over-write. (Maximum 5 sentences)

   The most simple exploit is code injection. The attacker should input more than 12 characters (I can't determine the exact number because of memory alignment issue) and overwrite the function return address area. The address that the attacker need to overwrite is 0xbffff820. By the way, the attacker can printf any stack data as he want.

4. What would you change to fix the problem in part 1?

   Please use C++ std::getline rather than unsafe scanf. An example written by me is here: https://github.com/recolic/rlib/blob/3a442c6dd8661d45cfe7528112b93c42ffa5d591/stdio.hpp#L52

   If I must figure out the implementation of std::getline, please read here: https://github.com/recolic/rlib/blob/3a442c6dd8661d45cfe7528112b93c42ffa5d591/sys/sio.hpp#L516

5. Given the code as is, would stack canaries prevent exploitation of this vulnerability?
   Select if yes ☐
   Why or why not?

Stack canaries can make the exploit harder, but it won't prevent the exploitation. The attacker can still printf data on stack. However, canaries are still very very useful to protect this program. I considered for some time and answer "no".

Problem 5 ***Reasoning About Memory Safety*** **(35 points)**

Consider the following C code.

```
1  /* (a) Precondition: _____ */
2  void dectohex(uint32_t decimal, char* hex) {
3      char tmp[9];
4      int digit, j = 0, k = 0;
5      do {
6          digit = decimal % 16;
7          if (digit < 10) {
8              digit += '0';
9          } else {
10             digit += 'A' - 10;
11         }
12         /* (b) Invariant: _____ */
13         tmp[j++] = digit;
14         decimal /= 16;
15     } while (decimal > 0);
16     while (j > 0) {
17         hex[k++] = tmp[--j];
18         /* (c) Invariant: _____ */
19     }
20     hex[k] = '\0';
21 }
```

1. Please identify the **preconditions** that must hold true for the following code to be memory safe. In addition, the precondition must be as conservative as possible (e.g. `decimal` cannot be required to be solely zero). Justify why your given precondition cannot be any less strict.

   Argument 'hex' must be a valid pointer to a writable memory space, and its size must be at least 9 bytes.

   If my precondition is not true, one of the following thing happens: 1. The pointer hex is invalid. It will write some random memory address or cause segmentation fault. 2. The buffer size is less than 9 bytes. Because 32bit unsigned integer has maximum value "0xffffffff", and we have k<=8. So buffer size must be at least 9 byte to avoid buffer overflow.

2. Please identify the loop **invariants** (b, c) that must hold true and justify them as well.

   b: j>=0 && j<=7

   c: j>=1 && j<=8, k>=0 && k<=7

Problem 6 ***Feedback*** **(0 points)**

Optionally, feel free to include feedback. What's the single thing 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 as your answer to Q6.