Popa & Wagner
Spring 2020

CS 161
Computer Security

Midterm

PRINT your name: ____________________________, ____________________________
(last) (first)

I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in partial or complete loss of credit.

SIGN your name: ____

PRINT your SID: ____

Name of the person sitting to your left: _________________________ Name of the person sitting to your right: _________________________

You may consult **one** double-sided, handwritten sheet of paper of notes. You may not consult other notes or textbooks. Calculators, computers, and other electronic devices are not permitted.

Bubble every item completely. Avoid using checkmarks or Xs.
If you want to unselect an option, erase it completely and clearly.

For questions with **circular bubbles**, you may select only one choice.

○ Unselected option (completely unfilled)

● Only one selected option (completely filled)

For questions with **square checkboxes**, you may select one or more choices.

■ You can select

■ multiple squares (completely filled).

If you think a question is ambiguous, please come up to the front of the exam room to the TAs. We will not answer your question directly. If we agree that the question is ambiguous we will add clarifications to the document projected in the exam rooms.

There is an appendix on the last page of the exam, containing all signatures of all C functions used on this exam and a synopsis. Please do not remove this appendix from the exam.

You have 80 minutes. There are 7 questions of varying credit (100 points total).

Do not turn this page until your instructor tells you to do so.

Problem 1 *Security Principles* (10 points)

Select the best answer to each question.

(a) A company requires that employees change their work machines' passwords every 30 days, but many employees find memorizing a new password every month difficult, so they either write it down or make small changes to existing passwords. Which security principle does the company's policy violate?

○ Defense in depth ○ Ensure complete mediation

○ Consider human factors ○ Fail-safe defaults

(b) In the midst of a PG&E power outage, Carol downloads a simple mobile flashlight app. As soon as she clicks a button to turn on the flashlight, the app requests permissions to access her phone's geolocation, address book, and microphone. Which security principle does this violate?

○ Security is economics ○ Least privilege

○ Separation of responsibility ○ Design in security from the start

(c) A private high school has 100 students, who each pay $10,000 in tuition each year. The principal hires a CS 161 alum as a consultant, who discovers that the "My Finances" section of the website, which controls students' tuition, is vulnerable to a brute force attack. The consultant estimates an attacker could rent enough compute power with $20 million to break the system, but tells the principal not to worry because of *which security principle?*

○ Security is economics ○ Design in security from the start

○ Least privilege ○ Consider human factors

(d) The consultant notices that a single admin password provides access to all of the school's funds and advises the principal that this is dangerous. What principle would the consultant argue the school is violating?

○ Don't rely on security through obscurity ○ Design security in from the start

○ Separation of responsibility ○ Fail-safe defaults

(e) Course staff at Stanford's CS155 accidentally released their project with solutions in it! In order to conceal what happened, they quickly re-released the project and didn't mention what had happened in the hope that no one would notice. This is an example of not following which security principle?

○ Security is economics ○ Know your threat model

○ Don't rely on security through obscurity ○ Least privilege

○ Separation of responsibility ○ None of these

Problem 2 ***Memory safety*** **(14 points)**

(a) TRUE or FALSE: In the last question of Project 1, ASLR prevents the attacker from knowing the address of any instructions in memory.

○ TRUE ○ FALSE

(b) TRUE or FALSE: An 8-byte stack canary is less secure than a 4-byte stack canary.

○ TRUE ○ FALSE

(c) Format string vulnerabilities can allow the attacker to:

□ Read memory □ Execute Shellcode

□ Write memory □ None of these

(d) Which of the following memory safety hardening measures work by ensuring that all writeable regions in memory are non-executable, and all executable regions in memory are non-writeable?

□ ASLR □ DEP (also known as WˆX or NX)

□ Stack canaries □ None of these

(e) Bear Systems hardens its code with both DEP (also known as WˆX or NX) and its own custom variant of ASLR. Normally, ASLR chooses a random offset for the stack and heap when the program starts running. Bear Systems modifies the compiler to choose a random offset when the program is compiled and hardcode this into the binary executable. Bear Systems ships the same executable to all of its customers. What is the effect of this modification to ASLR on security against memory safety exploits?

○ This modification makes security better.

○ This modification has no significant effect on security.

○ This modification makes security worse.

Problem 3 ***Symmetric-key Cryptography*** **(16 points)**

(a) True or False: AES-CBC mode requires both the sender and recipient to know the secret key and IV before communication begins.

○ True ○ False

(b) True or False: AES-CTR mode with a non-repeating but predictable IV is IND-CPA secure.

○ True ○ False

(c) True or False: AES-CBC mode with a non-repeating but predictable IV is IND-CPA secure.

○ True ○ False

(d) True or False: AES-ECB mode is IND-CPA secure if we prepend a random 16-byte value to the message before encryption and then encrypt the whole thing.

○ True ○ False

Consider the following modified version of CTR mode:

$$C_i = \text{AES}_K(P_i \oplus (IV \| i))$$

where || denotes concatenation and ⊕ denotes bitwise xor. In other words, the xor occurs **before** applying the block cipher. As always, assume that the IV is sent with the ciphertext.

(e) True or False: If the IV is different for each message but predictable, this mode is IND-CPA secure.

○ True ○ False

A cryptography consultant suggests the following alternative mode:

$$C_i = \text{AES}_K(P_i) \oplus (IV \| i)$$

where || denotes concatenation and ⊕ denotes bitwise xor. In other words, the IV and counter are xored to the output of the block cipher. As always, assume that the IV is sent with the ciphertext.

(f) True or False: If the IV is chosen randomly for each message, the consultant's mode is IND-CPA secure.

○ True ○ False

Problem 4 ***Software Vulnerabilities*** **(11 points)**

Consider the following C code:

```
// requires: s is a valid pointer, len <= size(s)
void f(char *s, size_t len) {
    int i, j;
    i=0; j=0;
    while (j < len) {
        // invariant: ???
        while (s[j] == '<')
            j++;
        s[i] = s[j];
        i++; j++;
    }
}
```

(a) Assume we will only ever call `f` with arguments where `s` is a valid, non-null pointer to a buffer of length at least `len`, and that the attacker controls the data stored in `s`. Is this code memory-safe, under those conditions?

○ Yes, it is memory-safe

○ No, it could write past the end of the buffer

○ No, it could read past the end of the buffer

○ No, it could write before the beginning of the buffer

○ No, it could read before the beginning of the buffer

(b) If you selected "Yes", write a valid loop invariant for the place marked `???`. If you selected "No", write an example value for `s` and `len` that would trigger a memory safety violation.

Invariant:

`s` =

`len` =

Problem 5 *Public Key Encryption* (7 points)

The El Gamal encryption scheme is reproduced below:

- **Key Generation**: public key = (g, h, p), where $h = g^k \pmod{p}$, private key = k
- **Encryption**: $c = (c_1, c_2) = (g^r \bmod p, m \times h^r \bmod p)$, where r is randomly sampled from $\{1, \dots, p-1\}$.
- **Decryption**: $m = c_1^{-k} \times c_2 \pmod{p}$

Look at each scenario below and select the appropriate options.

(a) TRUE or FALSE: With El Gamal, it is not a problem if the adversary can learn the value of g somehow.

○ TRUE ○ FALSE

(b) TRUE or FALSE: With El Gamal, it is not a problem if the value r used during encryption is accidentally revealed after the encryption is complete.

○ TRUE ○ FALSE

Problem 6 ***Block Cipher Leakage*** **(16 points)**

A hospital keeps a record, for each patient, of the patient's diseases. It is stored as a list of diseases along with a boolean indicating whether the patient has that disease or not:

```
acatamathesia: 0;ear infection: 0;heart disease: 1;...;xerophthalmia: 1;
```

Each record is encrypted. Assume that each "disease name: 0;" is exactly 16 bytes long (one block), disease names are all unique, and the list and order of diseases is public and the same for all patients.

A passive eavesdropper Eve intercepts two ciphertexts corresponding to the encryptions of Alice's and Bob's records. Assume that Eve has no prior knowledge of the disease status of any of the hospital's patients. The hospital uses the same key and **same IV** for encrypting each record.

(a) If the hospital uses AES-CBC mode with the same IV for every record, which of the following are true?

☐ Mallory can learn every disease for which Alice's boolean is equal to Bob's boolean

☐ Mallory can learn every disease for which Alice's boolean is not equal to Bob's boolean

☐ Mallory can always learn one disease for which Alice's boolean is equal to Bob's boolean, if any such disease exists

☐ Mallory can always learn one disease for which Alice's boolean is not equal to Bob's boolean, if any such disease exists

☐ Mallory can never learn two diseases for which Alice's boolean is equal to Bob's boolean

☐ Mallory can never learn two diseases for which Alice's boolean is not equal to Bob's boolean

☐ Mallory can learn whether Alice and Bob have the same boolean for all diseases

☐ Mallory cannot learn anything about Alice and Bob's booleans

(b) If the hospital uses AES-CTR mode with the same IV for every record, which are true?

☐ Mallory can learn every disease for which Alice's boolean is equal to Bob's boolean

☐ Mallory can learn every disease for which Alice's boolean is not equal to Bob's boolean

☐ Mallory can always learn one disease for which Alice's boolean is equal to Bob's boolean, if any such disease exists

☐ Mallory can always learn one disease for which Alice's boolean is not equal to Bob's boolean, if any such disease exists

☐ Mallory can never learn two diseases for which Alice's boolean is equal to Bob's boolean

☐ Mallory can never learn two diseases for which Alice's boolean is not equal to Bob's boolean

☐ Mallory can learn whether Alice and Bob have the same boolean for all diseases

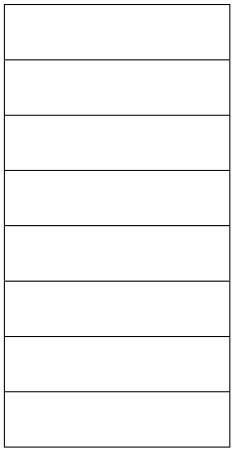
☐ Mallory cannot learn anything about Alice and Bob's booleans

Problem 7 ***Memory safety exploits*** **(26 points)**

The following code allows you to print characters of your choice from a string. It runs on a 32-bit x86 system with **stack canaries enabled**, but no other memory defense methods in use. Assume local variables are pushed onto the stack in the order that they are declared, and there is no extra padding, saved registers, or exception handlers. (These are the same assumptions as in homework 1.) Note that `scanf("%d", &offset)` reads a number from the input, converts it to an integer, and stores it in the `offset` variable.

```
 1 void foo() {
 2   char buf[300];
 3   gets(buf);
 4 }
 5 
 6 int main() {
 7   char *ptr;
 8   int offset = 0;
 9   char important[12] = "sEcuRitY!!!";
10   while (offset >= 0) {
11     scanf("%d", &offset);
12     ptr = important + offset;
13     printf("%c\n", *ptr);
14   }
15   foo();
16   return 0;
17 }
```

(a) Draw the stack, when at the point in time when line 12 of the code is executing, by filling in the diagram below. Label the location of sfp, rip (saved return address), stack canary, and the `ptr`, `offset`, and `important` variables, for `main`'s stack frame. Each empty box represents 4 bytes of stack memory. If a value spans multiple boxes, label all of them.



(b) Peyrin informs you that this code contains a vulnerability which leaks the value of `main`'s stack canary. Which sequence of inputs would leak this information? Fill in the blanks below.

_______ \n _______ \n _______ \n _______ \n

(c) Next, suppose you want to develop a reliable arbitrary-code-execution exploit that works by overwriting `foo`'s entire return address, so that when `foo` returns, your shellcode will be executed. You first supply the string from part (b) to learn the value of the stack canary, followed by the string '-1\n', followed by a carefully chosen third string of some length. Write the *minimum* possible length of the third string, to achieve this. Assume your shellcode is 100 bytes long and it cannot be shortened.

(d) Your friend claims that it's not necessary to overwrite the entire return address to achieve arbitrary code execution: if you don't get unlucky with where certain addresses happen to fall, it's possible to reduce the length of the third string in part (c) to 304 bytes or 305 bytes, using an exploit that overwrites the least significant byte of sfp. Is she right?

○ Yes　　○ No

(e) The developers propose to fix the program by replacing lines 12–13 with the following code. Fill in the blank inside the if-statement to make the fix correct.

```
12      if (____________________________________) {
13        ptr = important + offset;
14        printf("%c\n", *ptr);
15      }
```

Unfortunately the fix isn't available yet. Unsettled by your exploit, the sysadmins **enable ASLR** for the stack and the heap as a temporary defense for the rest of this question.

You discover that the code (text segment) is not randomized, and you learn the address of a `ret` instruction. For the purpose of this question, you can assume that `ret` is a one-word instruction which is equivalent to `pop %eip`. In other words, it loads the instruction at `$esp` into the `$eip` and increments `$esp` by one word.

(f) Which exploit technique would be appropriate for an arbitrary code execution exploit against this code, given this new information?

○ ROP　　○ Overwrite the first byte of sfp

○ TOCTTOU　　○ Exploit a format string vulnerability

(g) Provide bounds on x, such that the input 'x\n' will cause `ptr` to point somewhere in the region where `buf` will appear.

____________ ≤ x ≤ ____________

(h) Your exploit constructs an input as follows: first supply the string from part (b) to learn the value of the stack canary, followed by the string 'x\n' (with x chosen somehow based on part (g)) to set `ptr` appropriately, followed by a carefully chosen third string that is composed from multiple pieces. Below, select all possibilities for how to choose the third string so that the shellcode will be executed with probability at least 1/2.

Assume SHELLCODE is a 100-byte string containing the shellcode you want to execute, CANARY is the 4-byte value of the canary (learned using the technique from part (a)), `gadget` is the 4-byte address of the `ret` instruction you found, and NOPSLED is a 200-byte string containing many NOP instructions. Beware that `gets` will replace the newline at the end of your third string with a null byte, so your exploit might need to deal with this.

1. First 300 bytes of the third string:

□ SHELLCODE * 3

□ NOPSLED + SHELLCODE

□ SHELLCODE + 'a' * 196 + CANARY

□ `gadget` * 75

2. Next 12 bytes of the third string:

○ `gadget` * 3

○ CANARY * 3

○ CANARY + `gadget`*2

○ `gadget` * 2 + CANARY

○ CANARY + 'a' * 4 + CANARY

○ CANARY * 2 + 'a' * 4

3. Next bytes of the third string: (fill in the blank with a Python expression; your expression may reference SHELLCODE, NOP, CANARY, `gadget`, and 'a's, though you won't need them all)

4. Final byte of the third string:
\n

Selected C Manual Pages

```
char *gets(char *s);

       gets()  reads  a  line from stdin into the buffer pointed to by
       s until either a terminating newline or EOF, which it replaces
       with a null byte ('\0').

int printf(const char *format, ...);

       The  functions  in the printf() family produce output according to
       a format.  The functions printf() and vprintf() write output to
       stdout, the standard output stream.

       The format specifier %c prints a single character: the argument
       is interpreted as a character and printed.

int scanf(const char *format, ...);

       The scanf() family of functions scans input according to format
       as described  below.  This format may contain conversion
       specifications; the results from such conversions, if any, are
       stored in the locations pointed to by the pointer arguments that
       follow format.

       The format specifier %d reads an integer, represented in decimal
       notation, and writes it to the location pointed to by the argument.
```