Weaver
Spring 2021

CS 161
Computer Security

Practice Midterm

Please take this practice midterm to familiarize yourself with our midterm format. Fill in your answers on the Practice Midterm Answer Sheet assignment on Gradescope.

For questions with **circular bubbles**, you may select exactly *one* choice on Gradescope.

◯ Unselected option

● Only one selected option

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

■ You can select

■ multiple squares

For questions with a **large box**, you need to write your answer in the text box on Gradescope.

There is an appendix on the last page of this exam, containing descriptions of all C functions used on this exam.

You have 2 weeks (110 minutes for the actual exam). There are 4 questions of varying credit (63 points total).

[NOTE: The Practice Midterm Gradescope assignment is untimed, but the real exam will use a timed assignment. See the "Sample Timed Answer Sheet" on Gradescope for an example.]

The Gradescope answer sheet assignment has a time limit of 120 minutes. Do not click "Start Assignment" until you're ready to start the exam. The password to decrypt the PDF is at the top of the answer sheet.

The exam is open note. You can use an unlimited number of handwritten cheat sheets, but you must work alone.

Clarifications will be posted at https://cs161.org/clarifications.

Q1 *MANDATORY – Honor Code* (1 point)

Read the following honor code and sign your name on your answer sheet. *Failure to do so will result in a grade of 0 for this exam.*

I understand that I may not collaborate with anyone else on this exam, or cheat in any way. 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.

Q2 *True/false* **(14 points)**

[NOTE: The first question on the exam will always be True/False.]

Each true/false is worth 2 points. This question has 7 subparts.

[NOTE: On Gradescope, every question will be labeled as being worth 1 point–you should ignore this. The real point values are on the exam PDF.]

Q2.1 TRUE or FALSE: If the discrete-log problem is broken (someone finds a way to efficiently calculate a given $g^a \bmod p$), ElGamal encryption is no longer secure.

◯ TRUE ◯ FALSE

Q2.2 TRUE or FALSE: Buffer overflows can occur on the stack and heap, but not in the static section of C memory.

◯ TRUE ◯ FALSE

Q2.3 TRUE or FALSE: If ASLR is enabled, leaking the address of a stack variable would give an attacker the address of heap variables.

◯ TRUE ◯ FALSE

Q2.4 TRUE or FALSE: All cryptographic hash functions are one-to-one functions.

◯ TRUE ◯ FALSE

Q2.5 TRUE or FALSE: A company requires users to have long, complicated passwords. As a result, some employees write down their passwords on sticky notes to remember them. This is an example of not following the "Security is Economics" security principle.

◯ TRUE ◯ FALSE

Q2.6 TRUE or FALSE: Enabling stack canaries, ASLR, and DEP prevents all buffer overflow attacks.

◯ TRUE ◯ FALSE

Q2.7 TRUE or FALSE: Coding in a memory-safe language prevents all buffer overflow attacks.

◯ TRUE ◯ FALSE

This is the end of Q2. Leave the remaining subparts of Q2 blank on Gradescope, if there are any. Proceed to Q3 on your answer sheet.

Q3 *IV-e got a question for ya* **(24 points)**

Determine whether each of the following schemes is IND-CPA secure. This question has 6 subparts.

Q3.1 (3 points) AES-CBC where the IV for message M is chosen as SHA-256(x) truncated to the first 128 bits. x is a predictable counter starting at 0 and incremented *per message*.

[NOTE: Your answer sheet has six answer choices for every subpart, but not every question will have six answer choices. For example, for Q3.1 here, you should not use options (C), (D), (E), and (F) on your answer sheet.]

- (A) Insecure
- (B) Secure
- (C) —
- (D) —
- (E) —
- (F) —

Q3.2 (3 points) AES-CTR where the nonce for message M is chosen as SHA-256(x) truncated to the first 128 bits. x is a predictable counter starting at 0 and incremented *per message*.

[NOTE: The answer choices on this subpart are circular bubbles, so you should only bubble in one option out of (G) and (H) on your answer sheet for Q3.2.]

- (G) Insecure
- (H) Secure
- (I) —
- (J) —
- (K) —
- (L) —

Q3.3 (6 points) AES-CBC where the IV for message M is chosen as HMAC-SHA256(k_2, M) truncated to the first 128 bits. The MAC key k_2 is distinct from the encryption key k_1.

Provide a short justification for your answer. *Enter your answer in the text box on Gradescope.*

[NOTE: This question has a large empty box and circular bubbles, so you should choose (A) or (B) and type your justification in the text box for Q3.3 on Gradescope.]

- (A) Insecure
- (B) Secure
- (C) —
- (D) —
- (E) —
- (F) —

Q3.4 (6 points) AES-CTR where the nonce for message M is chosen as HMAC-SHA256(k_2, M) truncated to the first 128 bits. The MAC key k_2 is distinct from the encryption key k_1.

Provide a short justification for your answer. *Enter your answer in the text box on Gradescope.*

- (G) Insecure
- (H) Secure
- (I) —
- (J) —
- (K) —
- (L) —

Q3.5 (3 points) AES-CBC where the IV for message M is chosen as HMAC-SHA256($k_2 + x, M$) truncated to the first 128 bits. The MAC key k_2 is distinct from the encryption key k_1 and x is a predictable counter starting at 0 and incremented *per message*.

- (A) Insecure
- (B) Secure
- (C) —
- (D) —
- (E) —
- (F) —

Q3.6 (3 points) AES-CTR where the IV for message M is chosen as HMAC-SHA256($k_2 + x, M$) truncated to the first 128 bits. The MAC key k_2 is distinct from the encryption key k_1 and x is a predictable counter starting at 0 and incremented *per message*.

Clarification made during the exam: You can assume that IV refers to the nonce for CTR mode.

- (G) Insecure
- (H) Secure
- (I) —
- (J) —
- (K) —
- (L) —

[NOTE: You may not need all blanks on the answer sheet for a question. You should leave Q3.7 and Q3.8 blank on your answer sheet.]

This is the end of Q3. Leave the remaining subparts of Q3 blank on Gradescope, if there are any. Proceed to Q4 on your answer sheet.

Q4 *steg* (24 points)

This question has 8 subparts.

Consider a new C function, `steg(char *s)`. It is similar to `gets`, but instead of writing to higher memory addresses, `steg` stores the user input by writing to lower memory addresses, starting at the memory address pointed to by `s`.

For example, if I call `steg(str)` and `&str = 0xdeadbeef`, and I type in `xyz` as input, the byte `x` will be stored at `0xdeadbeef`, the byte `y` will be stored at `0xdeadbeee`, and the byte `z` will be stored at `0xdeadbeed`.

Consider the following vulnerable C code that uses `steg`:

```c
void display(char *buf) {
  steg(buf);
  printf("%s", buf);
}

int main() {
  char door[4];
  display(&door);
}
```

Fill in the numbered blanks for this incomplete stack diagram. Each box in the diagram represents 4 bytes. Each blank is worth 3 points.

rip of `main`
sfp of `main`
(1)
(2)
(3)
sfp of `display`

Q4.1 Blank (1)

- ◯ (A) `door`
- ◯ (B) `buf = &door`
- ◯ (C) rip of `display`
- ◯ (D) ——
- ◯ (E) ——
- ◯ (F) ——

Q4.2 Blank (2)

- ◯ (G) `door`
- ◯ (H) `buf = &door`
- ◯ (I) rip of `display`
- ◯ (J) ——
- ◯ (K) ——
- ◯ (L) ——

Q4.3 Blank (3)

- (A) `door`
- (B) `buf = &door`
- (C) rip of `display`
- (D) ——
- (E) ——
- (F) ——

Q4.4 (3 points) Which rip is vulnerable to being changed during the call to `steg`? Select all that apply.

Remember that the rip of a function `f` refers to the EIP of the previous function that is pushed onto the stack when calling `f`.

[NOTE: The answer choices on this subpart are square bubbles, so you should bubble in any of options (G) and (H) you think are correct, or only option (I) if you think all options are incorrect.You should leave options (J), (K), (L) on your answer sheet blank.]

- ☐ (G) `display`
- ☐ (H) `main`
- ☐ (I) None of the above
- ☐ (J) ——
- ☐ (K) ——
- ☐ (L) ——

Suppose we have an **8-byte** shellcode. Denote `REV_SHELLCODE` as a reversed version of this shellcode.

We find the address of `door` to be `0xbfffff1c`. Complete the exploit in the following three parts to cause the shellcode to execute.

Hint: x86 is little-endian (ie. the least significant byte of a word is stored at the lowest address), and we are writing from higher addresses to lower addresses.

Hint: `0xbfffff1c - 16 = 0xbfffff0c`, *and* `0xbfffff1c - 8 = 0xbfffff14`.

Q4.5 (3 points) At the call to `steg` at line 2, first input this many bytes of garbage to reach the rip:

- (A) 0
- (B) 1
- (C) 5
- (D) 9
- (E) 13
- (F) 17

Q4.6 (3 points) Then overwrite the rip with these bytes:

- (G) `\xbf\xff\xff\x0c`
- (H) `\x0c\xff\xff\xbf`
- (I) `\xbf\xff\xff\x14`
- (J) `\x14\xff\xff\xbf`
- (K) `REV_SHELLCODE`
- (L) ——

Q4.7 (3 points) Then input these bytes:

- (A) `\xbf\xff\xff\x0c`
- (B) `\x0c\xff\xff\xbf`
- (C) `\xbf\xff\xff\x14`
- (D) `\x14\xff\xff\xbf`
- (E) `REV_SHELLCODE`
- (F) ——

Q4.8 (3 points) Would the exploit from the previous parts still work if stack canaries were enabled? Assume there is no way for the attacker to learn the value of the stack canary.

○ (G) Yes ○ (H) No ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

This is the end of Q4. Leave the remaining subparts of Q4 blank on Gradescope, if there are any. You have reached the end of the exam.