# Popa and Weaver
# Fall 2021

# CS 161
# Computer Security

# Midterm

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

PRINT your student ID: _____

You have 110 minutes. There are 8 questions of varying credit (150 points total).

---

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).

---

Pre-exam activity (not graded, just for fun): Draw lines to match each character to their name.

Alice Bob EvanBot Eve Mallory

---

## Q1 *Honor Code* **(4 points)**

**Read the following honor code and sign your name.**

> 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, at minimum, negative points on the exam and a corresponding notch on Nick's Stanley Fubar demolition tool.

SIGN your name: _____

## Q2 *True/false* **(30 points)**

Each true/false is worth 2 points.

Q2.1 TRUE or FALSE: Assume stack canaries, non-executable pages, and pointer authentication codes are enabled, but ASLR is disabled. An attacker can execute the RET2ESP attack from Project 1, Question 6.

◯ TRUE ◯ FALSE

Q2.2 TRUE or FALSE: Assume that ASLR is enabled, but all other memory safety defenses are disabled. It's possible to find an exploit that will allow an attacker to learn the address of the RIP during program execution.

◯ TRUE ◯ FALSE

Q2.3 TRUE or FALSE: When designing a secure system, we often assume that an adversary has access to our source code.

◯ TRUE ◯ FALSE

Q2.4 TRUE or FALSE: If pointer authentication codes are enabled, the program will always crash if the user writes past the end of a buffer.

◯ TRUE ◯ FALSE

Q2.5 TRUE or FALSE: Let $\mathsf{Enc}(K, M)$ be an IND-CPA secure encryption function. If Alice computes $\mathsf{Enc}$("Hello", "World") and Bob computes $\mathsf{Enc}$("Hello", "World"), they will always evaluate to the same ciphertext.

◯ TRUE ◯ FALSE

Q2.6 A PRNG has been securely initialized with truly random bits. Assume the attacker has not learned the PRNG's internal state.

TRUE or FALSE: It is infeasible for the attacker to distinguish $n$ bits of PRNG output from $n$ truly random bits.

◯ TRUE ◯ FALSE

Q2.7 Alice and Bob are building a password management scheme for CalCentral. Carol tells them that they should store a record of $(\mathsf{username}, \mathsf{MD5}(\mathsf{password}))$ for each user in the database.

TRUE or FALSE: This design protects against offline password hashing attacks.

◯ TRUE ◯ FALSE

Q2.8 TRUE or FALSE: When designing a password management scheme, all security guarantees are lost if the attacker gains access to a table containing salt values.

◯ TRUE ◯ FALSE

Q2.9 TRUE or FALSE: Assume that stack canaries are enabled, but all other memory safety defenses are disabled. A format string vulnerability in a vulnerable C program may allow an attacker to write arbitrary bytes anywhere on the stack.

◯ TRUE ◯ FALSE

Q2.10 TRUE or FALSE: If a website requires all passwords to be at least 20 characters long, then it's more difficult for an adversary to perform rainbow table attacks on unsalted password hashes.

○ TRUE ○ FALSE

Q2.11 TRUE or FALSE: Diffie-Hellman is susceptible to man-in-the-middle attacks.

○ TRUE ○ FALSE

Q2.12 Bob has public key $PK_B$ and private key $SK_B$. Assume that Sign is a secure digital signature algorithm.

TRUE or FALSE: $\{\text{"Bob's public key is } PK_B\text{"}\}_{SK_B^{-1}}$ is a valid certificate on Bob's public key.

○ TRUE ○ FALSE

Q2.13 Recall that in ElGamal, the ciphertext is $C_1 = g^r \bmod p$ and $C_2 = M \times (g^b)^r \bmod p$.

TRUE or FALSE: Alice encrypts $M$ and sends the corresponding ciphertext $C_1$ and $C_2$ to Bob. If Mallory replaces the ciphertext with $2 \times C_1$ and $2 \times C_2$, Bob will decrypt the ciphertext as $2 \times M$.

○ TRUE ○ FALSE

Q2.14 Alice and Bob share a symmetric key $K$ not known to anyone else, and MAC is a secure (unforgeable) MAC scheme. Alice presents the statement $M =$ "Bob owes Alice 100 dollars" and the MAC $\text{MAC}(K, M)$ to a judge.

TRUE or FALSE: The judge can be sure that the message was sent by Bob to Alice.

○ TRUE ○ FALSE

Q2.15 TRUE or FALSE: In the Bitcoin ledger, everyone can add new entries and modify/delete existing entries.

○ TRUE ○ FALSE

## Q3 *EvanBot Alpha* **(15 points)**

Earlier releases of EvanBot sometimes came with security vulnerabilities. For each subpart, select the most relevant security principle. Each option is used exactly once.

Q3.1 (3 points) When debugging students' code, EvanBot Alpha ran all code with administrator privileges. The latest version of EvanBot runs untrusted code in an isolated environment (sandbox) instead.

- ◯ Consider human factors
- ◯ Least privilege
- ◯ Separation of responsibility
- ◯ Don't rely on security through obscurity
- ◯ Design in security from the start

Q3.2 (3 points) Any TA by themselves could instruct EvanBot Alpha to release exam solutions. The latest version of EvanBot requires two TAs to approve the instruction before releasing exam solutions.

- ◯ Consider human factors
- ◯ Least privilege
- ◯ Separation of responsibility
- ◯ Don't rely on security through obscurity
- ◯ Design in security from the start

Q3.3 (3 points) The default setting for EvanBot Alpha was to store exam solutions with no encryption. To encrypt exam solutions with AES, TAs had to manually change a setting in EvanBot Alpha's source code. The latest version of EvanBot encrypts exam solutions by default.

- ◯ Consider human factors
- ◯ Least privilege
- ◯ Separation of responsibility
- ◯ Don't rely on security through obscurity
- ◯ Design in security from the start

Q3.4 (3 points) To protect exam solutions, EvanBot Alpha used a secret encryption algorithm. The latest version of EvanBot uses a well-known encryption algorithm with a secret key.

- ◯ Consider human factors
- ◯ Least privilege
- ◯ Separation of responsibility
- ◯ Don't rely on security through obscurity
- ◯ Design in security from the start

Q3.5 (3 points) EvanBot Alpha required frequent patches to fix vulnerabilities. The latest version of EvanBot's source code was rewritten from scratch and requires far fewer security patches.

- ◯ Consider human factors
- ◯ Least privilege
- ◯ Separation of responsibility
- ◯ Don't rely on security through obscurity
- ◯ Design in security from the start

## Q4 *CIA* (15 points)

Alice and Bob want to communicate securely over an insecure channel. Mallory can read and modify messages sent over the channel. Determine whether each scheme provides confidentiality, integrity, both, or neither. Each subpart is independent.

Assumptions:

- ECB and CBC denote AES-ECB and AES-CBC encryption, respectively.
- MAC is a secure (existentially unforgeable) MAC scheme.
- Sign denotes RSA signatures.
- Alice and Bob share two symmetric keys, $K_1$ and $K_2$, not known to anyone else.
- Alice has an RSA key pair $(PK_a, SK_a)$. Bob knows $PK_a$, and $SK_a$ is not known to anyone other than Alice.

Q4.1 (3 points) Alice sends over $\mathsf{CBC}(K_1, M)$ and $\mathsf{MAC}(K_2, M)$.

◯ Confidentiality ◯ Both

◯ Integrity ◯ Neither

Q4.2 (3 points) Alice computes $C = \mathsf{CBC}(K_1, M)$. Then, Alice sends $\mathsf{ECB}(K_2, C)$ and $\mathsf{Sign}(SK_a, C)$.

◯ Confidentiality ◯ Both

◯ Integrity ◯ Neither

Q4.3 (3 points) Alice computes $C_0 = \mathsf{CBC}(K_1, M)$ and $C_1 = \mathsf{ECB}(K_1, M)$. Then, Alice sends $C_0$, $C_1$, and $\mathsf{Sign}(SK_a, C_0)$.

◯ Confidentiality ◯ Both

◯ Integrity ◯ Neither

Q4.4 (3 points) Alice computes $C = \mathsf{CBC}(K_1, M)$. Alice sends $C$ and $\mathsf{MAC}(K_2, C)$.

◯ Confidentiality ◯ Both

◯ Integrity ◯ Neither

Q4.5 (3 points) Alice sends $\mathsf{CBC}(K_1, M)$ and $\mathsf{Sign}(SK_a,$ "This message was sent by Alice").

◯ Confidentiality ◯ Both

◯ Integrity ◯ Neither

## Q5 *Ivy Why* **(21 points)**

Alice wants to send a 400-bit message $M$ to Bob. Assume that Alice is using a block cipher with 128-bit blocks and that Alice is using PKCS#7 for padding (this is the padding scheme from Homework 2).

Eve observes the encryption of $M$. Assume that Eve knows which encryption scheme is being used. For each encryption scheme, what is the most specific information Eve can learn about $\mathsf{len}(M)$, the length of $M$ in bits?

Q5.1 (3 points) Alice pads the plaintext to the nearest multiple of the block size and then encrypts the padded plaintext with CBC mode.

- ◯ $\mathsf{len}(M) = 400$
- ◯ $256 \leq \mathsf{len}(M) < 384$
- ◯ $512 \leq \mathsf{len}(M)$
- ◯ $\mathsf{len}(M) < 256$
- ◯ $384 \leq \mathsf{len}(M) < 512$
- ◯ None of the above

Q5.2 (3 points) Alice pads the plaintext to the nearest multiple of the block size and then encrypts the padded plaintext with CTR mode.

- ◯ $\mathsf{len}(M) = 400$
- ◯ $256 \leq \mathsf{len}(M) < 384$
- ◯ $512 \leq \mathsf{len}(M)$
- ◯ $\mathsf{len}(M) < 256$
- ◯ $384 \leq \mathsf{len}(M) < 512$
- ◯ None of the above

Q5.3 (3 points) Alice encrypts the message with CTR mode and then pads the ciphertext to the nearest multiple of the block size.

- ◯ $\mathsf{len}(M) = 400$
- ◯ $256 \leq \mathsf{len}(M) < 384$
- ◯ $512 \leq \mathsf{len}(M)$
- ◯ $\mathsf{len}(M) < 256$
- ◯ $384 \leq \mathsf{len}(M) < 512$
- ◯ None of the above

For the rest of this question, Alice is sending two 400-bit messages $M_1$ and $M_2$ to Bob. $M_1$ and $M_2$ are identical except in the 200th bit.

Eve observes the encryption of $M_1$ and the encryption of $M_2$. Assume that Eve knows which encryption scheme is being used. For each encryption scheme, what is the most specific information that Eve can learn about how the messages differ? Assume that inputs for block cipher modes that require padding are correctly padded.

Q5.4 (3 points) Alice encrypts the messages with ECB mode.

- ◯ The messages differ in only the 200th bit
- ◯ The messages differ in only the second block (Eve doesn't know which bit)
- ◯ The messages differ in the second block and possibly subsequent blocks
- ◯ The messages differ somewhere (Eve doesn't know which blocks)
- ◯ None of the above

Q5.5 (3 points) Alice encrypts the messages with CBC mode. Alice uses the same IV for both encryptions.

- ◯ The messages differ in only the 200th bit
- ◯ The messages differ in only the second block (Eve doesn't know which bit)
- ◯ The messages differ in the second block and possibly subsequent blocks
- ◯ The messages differ somewhere (Eve doesn't know which blocks)
- ◯ None of the above

Q5.6 (3 points) Alice encrypts the messages with CTR mode (with no padding). Alice uses the same nonce for both encryptions.

- ◯ The messages differ in only the 200th bit
- ◯ The messages differ in only the second block (Eve doesn't know which bit)
- ◯ The messages differ in the second block and possibly subsequent blocks
- ◯ The messages differ somewhere (Eve doesn't know which blocks)
- ◯ None of the above

Q5.7 (3 points) Alice encrypts the messages with CTR mode (with no padding). Alice uses a different, random nonce for each encryption.

- ◯ The messages differ in only the 200th bit
- ◯ The messages differ in only the second block (Eve doesn't know which bit)
- ◯ The messages differ in the second block and possibly subsequent blocks
- ◯ The messages differ somewhere (Eve doesn't know which blocks)
- ◯ None of the above

## Q6 *Bonsai* (22 points)

EvanBot wants to store a file in an *untrusted* database that the adversary can read and modify.

Before storing the file, EvanBot computes a hash over the contents of the file and stores the hash separately. When retrieving the file, EvanBot re-computes a hash over the file contents, and, if the computed hash doesn't match the stored hash, then EvanBot concludes that the file has been tampered with.

Q6.1 (4 points) What assumptions are needed for this scheme to guarantee integrity on the file? Select all that apply.

- ☐ An attacker cannot tamper with EvanBot's stored hash
- ☐ EvanBot has a secret key that nobody else knows
- ☐ The file is at most 128 bits long
- ☐ EvanBot uses a secure cryptographic hash
- ☐ None of the above

For the rest of this question, we refer to two databases: a *trusted database* that an adversary cannot read or modify, and an *untrusted database* that an adversary can read and modify.

Assume that $\mathsf{H}$ is a secure cryptographic hash function and $\|$ denotes concatenation.

EvanBot creates and stores four files, $F_1$, $F_2$, $F_3$, and $F_4$, in the untrusted database. EvanBot also computes and stores a hash on each file's contents in the untrusted database:

$$h_1 = \mathsf{H}(F_1) \qquad h_2 = \mathsf{H}(F_2) \qquad h_3 = \mathsf{H}(F_3) \qquad h_4 = \mathsf{H}(F_4)$$

Then, EvanBot stores $h_{\mathsf{root}} = \mathsf{H}(h_1 \| h_2 \| h_3 \| h_4)$ in the *trusted* database.

Q6.2 (3 points) If an attacker modifies $F_2$ stored on the server, will EvanBot be able to detect the tampering?

- ◯ Yes, because EvanBot can compute $h_{\mathsf{root}}$ and see it doesn't match the stored $h_{\mathsf{root}}$
- ◯ Yes, because EvanBot can compute $h_2$ and see it doesn't match the stored $h_2$
- ◯ No, because the hash doesn't use a secret key
- ◯ No, because the attacker can re-compute $h_2$ to be the hash of the modified file

Q6.3 (3 points) What is the minimum number of hashes EvanBot needs to compute to verify the integrity of all four files?

| | | |
|---|---|---|
| ◯ 1 | ◯ 3 | ◯ 5 |
| ◯ 2 | ◯ 4 | ◯ More than 5 |

For the rest of the question, assume that none of the other files besides $F_2$ and none of the hashes have been modified.

Q6.4 (3 points) What is the minimum number of hashes EvanBot needs to compute to verify the integrity of only $F_2$?

- ◯ 1
- ◯ 3
- ◯ 5
- ◯ 2
- ◯ 4
- ◯ More than 5

Q6.5 (3 points) If EvanBot uses the minimum number of hash computations, how many hashes from the untrusted database are used to verify the integrity of only $F_2$?

- ◯ 1
- ◯ 3
- ◯ 5
- ◯ 2
- ◯ 4
- ◯ More than 5

EvanBot changes the protocol as follows:

Before storing the files, EvanBot first computes the hash of each file individually and stores the following in the *untrusted* database:

$$h_1 = \mathsf{H}(F_1) \qquad h_2 = \mathsf{H}(F_2) \qquad h_3 = \mathsf{H}(F_3) \qquad h_4 = \mathsf{H}(F_4)$$

Then, EvanBot computes and stores the following in the *untrusted* database:

$$h_{\mathsf{left}} = \mathsf{H}(h_1||h_2)$$

$$h_{\mathsf{right}} = \mathsf{H}(h_3||h_4)$$

Finally EvanBot computes and stores the following in the *trusted* database:

$$h_{\mathsf{root}} = \mathsf{H}(h_{\mathsf{left}}||h_{\mathsf{right}})$$

Q6.6 (3 points) What is the minimum number of hashes EvanBot needs to compute to verify the integrity of only $F_2$?

- ◯ 1
- ◯ 3
- ◯ 5
- ◯ 2
- ◯ 4
- ◯ More than 5

Q6.7 (3 points) If EvanBot uses the minimum number of hash computations, how many hashes from the untrusted database are used to verify the integrity of only $F_2$?

- ◯ 1
- ◯ 3
- ◯ 5
- ◯ 2
- ◯ 4
- ◯ More than 5

## Q7 *Returnless* (20 points)

Consider the following vulnerable C code:

```
1  int main(void) {
2      int i;
3      char input[16];
4      int stored_nums[4];
5
6      for (i = 0; i < 4; i++) {
7          printf("Enter stored number for %d: ", i);
8          gets(input);
9          stored_nums[i] = atoi(input);
10     }
11
12     while (i >= 0) {
13         printf("Which stored number do you want to print? ");
14         gets(input);
15         i = atoi(input);
16         if (i >= 0) {
17             printf("%d\n", stored_nums[i]);
18         }
19     }
20     return 0;
21 }
```

Assume you are on a little-endian 32-bit x86 system. Assume that there is no compiler padding or additional saved registers in all subparts. Assume that **stack canaries are enabled**, and all other memory safety defenses are disabled (unless otherwise specified).

Q7.1 (3 points) Assume the program is paused immediately after executing line 4. Complete the stack diagram below.

| (a) |
|---|
| SFP of `main` |
| (b) |
| (c) |
| (d) |
| (e) |

- ◯ (a) - canary; (b) - RIP of `main`; (c) - `input`; (d) - `stored_nums`; (e) - `i`
- ◯ (a) - canary; (b) - `i`; (c) - `input`; (d) - `stored_nums`; (e) - RIP of `main`
- ◯ (a) - RIP of `main`; (b) - canary; (c) - `i`; (d) - `input`; (e) - `stored_nums`
- ◯ (a) - RIP of `main`; (b) - canary; (c) - `stored_nums`; (d) - `input`; (e) - `i`
- ◯ (a) - canary; (b) - `printf`; (c) - `stored_nums`; (d) - `input`; (e) - `i`
- ◯ (a) - canary; (b) - `i`; (c) - `input`; (d) - `stored_nums`; (e) - `printf`

Q7.2 (3 points) Which of the following vulnerabilities is present in this code?

- ◯ Buffer overflow
- ◯ Format string vulnerability
- ◯ Integer overflow vulnerability
- ◯ Signed/unsigned vulnerability
- ◯ None of the above

Q7.3 (4 points) Which of these inputs to the `gets` call at line 14 will leak the value of the canary? Select all that apply.

- ☐ `'-1'`
- ☐ `'\x00'`
- ☐ `'9'`
- ☐ `'12'`
- ☐ None of the above

Q7.4 (4 points) Assume that the address of `input` is `0x7ffc9180`.

Fill in the blanks to construct another input to the `gets` call at line 14 that will overwrite the canary with itself and cause the program to execute malicious shellcode.

Write your answer in Python 2 syntax (just like Project 1). You can use the variable `CANARY` as the leaked 4-byte canary value and the variable `SHELLCODE` as the 32-byte shellcode.

```
'____________' + '\x_____' + ('A' * _____) + ________________ +

('B' * _____) + --'\x_____\x_____\x_____\x_____' + ________________ + '\n'
```

Q7.5 (3 points) Is it possible to exploit this program without overwriting the stack canary (even with itself)?

- ◯ Yes, using a format string vulnerability at line 7
- ◯ Yes, using a write at line 9
- ◯ Yes, using a format string vulnerability at line 13
- ◯ Yes, using the call to `gets` at line 14
- ◯ No, because you can't learn the address of the RIP
- ◯ No, because stack canaries prevent the value of the RIP from changing at all

Q7.6 (3 points) For this subpart only, assume that the code is run on a 64-bit system with stack canaries disabled but pointer authentication enabled.

Assume that you have found a vulnerability that would allow the used bits of a 64-bit address to be overwritten without touching the bits used by the PAC. Would this vulnerability, by itself, allow an attacker to execute malicious shellcode?

- ◯ Yes, because this vulnerability leaves the PAC bits undisturbed
- ◯ Yes, because this vulnerability overwrites the PAC bits with itself
- ◯ No, because the value of the PAC depends on the address of the pointer
- ◯ No, because the PAC is deterministic

## Q8 *161 Meets 61A* **(23 points)**

Consider the following buggy C code:

```
1  void add_letter(int i, char *buf) {
2      char word[4];
3      printf("Enter Word %d:\n", i);
4      fgets(word, 4, stdin);
5      buf[i] = word[0];
6      if (i > 0) {
7          add_letter(i - 1, buf);
8      }
9  }
10
11 void make_acronym(void) {
12     char result[4];
13     add_letter(4, result);
14     printf("%s\n", result);
15 }
16
17 void word_games(void) {
18     make_acronym();
19 }
20
21 int main(void) {
22     word_games();
23     return 0;
24 }
```

Assume you are on a little-endian 32-bit x86 system. Assume that there is no compiler padding or additional saved registers in all subparts. Assume all memory-safety defenses (ASLR, stack canaries, pointer authentication codes, and non-executable pages) are disabled, unless otherwise specified.

Q8.1 (3 points) How many times will the `add_letter` function be run each time the `make_acronym` function is called?

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5

Q8.2 (4 points) Which value(s) will be overwritten (partially or completely) when you provide an input for the prompt to "Enter Word 4:"? Select all that apply.

☐ RIP of `word_games`

☐ SFP of `word_games`

☐ RIP of `make_acronym`

☐ SFP of `make_acronym`

☐ None of the above

Assume that malicious shellcode is stored at `0x44332211` and the address of `result` is `0xAABBCCB8`. In the next five subparts, provide a series of inputs to `fgets` that would cause the program to execute shellcode.

Q8.3 (1 point) First input:

◯ `\xA9` ◯ `\xAC` ◯ `\xB0` ◯ `\xB4` ◯ `\xB8` ◯ `\xBC`

Q8.4 (1 point) Second input:

◯ `\x00` ◯ `\x11` ◯ `\x22` ◯ `\x33` ◯ `\x44` ◯ `\x48`

Q8.5 (1 point) Third input:

◯ `\x00` ◯ `\x11` ◯ `\x22` ◯ `\x33` ◯ `\x44` ◯ `\x48`

Q8.6 (1 point) Fourth input:

◯ `\x00` ◯ `\x11` ◯ `\x22` ◯ `\x33` ◯ `\x44` ◯ `\x48`

Q8.7 (1 point) Fifth input:

◯ `\x00` ◯ `\x11` ◯ `\x22` ◯ `\x33` ◯ `\x44` ◯ `\x48`

Q8.8 (3 points) Assume that you've successfully executed the exploit above. At what point will the function jump to your shellcode?

- ◯ When `main` returns
- ◯ When `word_games` returns
- ◯ When `make_acronym` returns
- ◯ When `add_letter` (called with `i == 4`) returns
- ◯ When `add_letter` (called with `i == 3`) returns
- ◯ None of the above

Q8.9 (3 points) **For this subpart only, assume stack canaries are enabled.** Suppose the CPU generates a different 4-byte canary for each function call by taking the SHA-512 hash of the RIP and using the first 4 bytes as the canary.

Assume the attacker can execute this program in GDB. Using this scheme, which canary values would an attacker would be able to learn? Select all that apply.

- ☐ `main`
- ☐ `make_acronym`
- ☐ `word_games`
- ☐ None of the above

Q8.10 (5 points) **For this subpart only, assume stack canaries are enabled.** Assume that the CPU generates a random four-byte canary, but the least-significant byte is always `0xAA`. Which series of inputs to `fgets` will cause the program to leak the value of the stack canary? Select all that apply.

- ☐ `\xAA`, `\xAA`, `\xAA`, `\xAA`, `\xAA`
- ☐ `\xAA`, `\x00`, `\xFF`, `\xFF`, `\xAA`
- ☐ `\xAA`, `\x11`, `\x22`, `\x33`, `\x44`
- ☐ `\xB8`, `\xFF`, `\xFF`, `\xFF`, `\x00`
- ☐ `\xB8`, `\xCC`, `\xFF`, `\xFF`, `\xAA`
- ☐ None of the above

# Doodle Page