Weaver Spring 2021 — CS 161 Computer Security — Midterm

For any logistics emergencies during the exam, email cs161-staff@berkeley.edu or text (252) 410-1123.

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 at the end of this exam, containing descriptions of all C functions used on this exam.

You have 110 minutes, plus a 10-minute buffer for distractions or technical difficulties, for a total of 120 minutes. There are 7 questions of varying credit (150 points total).

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* (5 points)

Read the following honor code and type your name on Gradescope.

> 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.

This is the end of Q1. Leave the remaining subparts of Q1 blank on Gradescope, if there are any. Proceed to Q2 on your answer sheet.

Q2 *True/false* **(40 points)**

Each true/false is worth 2 points.

Q2.1 Consider the following vulnerable function:

```
1 void vulnerable() {
2     char buf[32];
3     gets(buf);
4     printf(buf);
5 }
```

TRUE or FALSE: Replacing `gets(buf)` with `fgets(buf, 32, stdin)` makes this function memory-safe.

◯ TRUE ◯ FALSE

Q2.2 TRUE or FALSE: In practice, El Gamal encryption is usually used to encrypt random session keys, not meaningful messages.

◯ TRUE ◯ FALSE

Q2.3 TRUE or FALSE: While using AES-CTR mode, nonces associated with future ciphertexts can be published ahead of time without breaking security.

◯ TRUE ◯ FALSE

Q2.4 TRUE or FALSE: When ASLR is enabled, it is possible to redirect to shellcode that is located on the stack.

◯ TRUE ◯ FALSE

Q2.5 TRUE or FALSE: A pseudorandom generator can be used to stretch an initial seed with k bits of entropy to a longer output with $2k$ bits of entropy.

◯ TRUE ◯ FALSE

Q2.6 TRUE or FALSE: Suppose p is a prime and g is a generator (just like in Diffie-Hellman). Given $g^a \pmod{p}$, an attacker with unlimited computational resources cannot recover a.

◯ TRUE ◯ FALSE

Q2.7 TRUE or FALSE: When W^X (also known as non-executable pages, DEP, or the NX bit) is enabled, memory on the heap can be interpreted as code and executed.

◯ TRUE ◯ FALSE

Q2.8 TRUE or FALSE: In general, we want our trusted computing base (TCB) to be as large as possible, in order to ensure that all components of a software system are trusted components.

◯ TRUE ◯ FALSE

Q2.9 TRUE or FALSE: A bank vault is protected by a locked door, but thieves break into the vault by entering the apartment upstairs and drilling a hole through the ceiling. This is an example of least privilege.

○ True ○ False

Q2.10 True or False: One-time pad encryption and decryption can both be parallelized.

○ True ○ False

Q2.11 True or False: Pointer authentication is a commonly-used defense on 32-bit systems.

○ True ○ False

Q2.12 True or False: If you browse the Internet through Tor, all your communications are guaranteed to be anonymous (no adversary can see who you're communicating with).

○ True ○ False

Q2.13 True or False: Format string vulnerabilities let us read values from memory, but not write to memory.

○ True ○ False

Q2.14 True or False: The fastest computers today are capable of brute-forcing a 128-bit key in about 20 years.

○ True ○ False

Q2.15 True or False: If the secret key is randomly generated for each encryption, then even if nonces are reused, AES-CTR mode is still IND-CPA secure.

○ True ○ False

Q2.16 True or False: While using AES-CBC mode, an IV associated with a ciphertext should never be revealed to an eavesdropper at any time.

○ True ○ False

Q2.17 True or False: Password hashing algorithms should use slower hashes.

○ True ○ False

Q2.18 True or False: Time-of-check to time-of-use (TOCTTOU) vulnerabilities can be present in memory-safe programming languages such as Python.

○ True ○ False

Q2.19 True or False: To solve a Bitcoin proof-of-work problem, a miner has to find a value whose hash begins with many zeros.

○ True ○ False

Q2.20 True or False: A program with ASLR, stack canaries, and W^X (also known as non-executable pages, DEP, or the NX bit) enabled is still vulnerable to integer conversion vulnerabilities.

○ 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 *Stonks* (22 points)

You are an engineer for the innovative^{TM} stock trading app Hobinrood, and you notice that after some recent market shenanigans, attacks on your users' accounts are through the roof. Hobinrood needs you to analyze the security of a few potential password storage schemes.

In this question, H is a secure cryptographic hash function, $\oplus$ denotes bitwise XOR, and $\|$ denotes concatenation.

The attacker in this question has access to the entire password database, but no access to any data not explicitly stored in the database. The database does not store any extra information besides what is listed in each subpart. Each subpart is independent.

Assume that there are n users who each choose from a common pool of n possible passwords, and the attacker has enough compute power to perform $O(n)$ hashes, decryptions, XORs, and other computations.

For each of the following password storage schemes, select all statements that are **guaranteed** to be true.

Q3.1 (3 points) For each user, the database stores $(\texttt{username}, H(\texttt{password}))$.

- ☐ (A) The attacker can learn all users' passwords.
- ☐ (B) The attacker can learn **at least one** user's password.
- ☐ (C) The attacker can determine all pairs of users who share the same password.
- ☐ (D) None of the above
- ☐ (E) ——
- ☐ (F) ——

Q3.2 (3 points) For each user, the database stores $(\texttt{username}, H(\texttt{username}) \oplus \texttt{password})$.

You can assume that the output of H is at least as long as the maximum password length.

- ☐ (G) The attacker can learn all users' passwords.
- ☐ (H) The attacker can learn **at least one** user's password.
- ☐ (I) The attacker can determine all pairs of users who share the same password.
- ☐ (J) None of the above
- ☐ (K) ——
- ☐ (L) ——

Q3.3 (3 points) For each user, the database stores $(\texttt{username}, r, H(\texttt{password}\|r))$.

r is a random 1024-bit value selected when the user creates their account.

- ☐ (A) The attacker can learn all users' passwords.

☐ (B) The attacker can learn **at least one** user's password.

☐ (C) The attacker can determine all pairs of users who share the same password.

☐ (D) None of the above

☐ (E) ——

☐ (F) ——

Q3.4 (3 points) For each user, the database stores $(\texttt{username}, \text{AES-CBC}(k, H(\texttt{password})))$.

AES-CBC denotes AES-CBC mode encryption, with a random, unpredictable IV used for each encryption. k is a secret key that the password database knows, but the attacker doesn't know.

☐ (G) The attacker can learn all users' passwords.

☐ (H) The attacker can learn **at least one** user's password.

☐ (I) The attacker can determine all pairs of users who share the same password.

☐ (J) None of the above

☐ (K) ——

☐ (L) ——

Q3.5 (3 points) Because usernames are often unique to a website, some websites opt to salt the password hash with the username rather than a random number. Consider storing $(\texttt{username}, H(\texttt{password}\|\texttt{username}))$ for each user. Briefly describe one disadvantage of this scheme compared to using random salts, i.e. storing $(\texttt{username}, r, H(\texttt{password}\|r))$.

Enter your answer in the text box on Gradescope.

◯ (A) —— ◯ (B) —— ◯ (C) —— ◯ (D) —— ◯ (E) —— ◯ (F) ——

You realize that designing a secure password storage scheme can be hard and decide to think about ways to let users log in without passwords.

Q3.6 (4 points) Which of the following protocols would allow you to verify a user's identity? Assume that you know the user's public key, and the user's private key has not been compromised. Select all that apply.

☐ (G) Perform Diffie-Hellman key exchange with the user to get a shared key k. The user tells you k. (You can assume no MITM has tampered with the key exchange.)

☐ (H) The user gives you a certificate with their public key, signed by a trustworthy certificate authority.

☐ (I) Encrypt a random value r with the user's public key. The user tells you r.

☐ (J) Give the user a random value r. The user signs r and sends you the signature.

☐ (K) None of the above

☐ (L) ——

Hobinrood uses a certificate hierarchy to validate users' public keys. In the hierarchy, a trusted certificate authority (CA) issues certificates to apps such as Hobinrood. Each app then issues certificates for its trusted users.

Q3.7 (3 points) An attacker shows you a valid certificate for the attacker's public key that appears to be signed by Hobinrood and a valid certificate for Hobinrood signed by the trusted CA. You know that Hobinrood would never issue a certificate to the attacker. What could the attacker have done to accomplish this? Select all that apply.

☐ (A) Stolen the CA's private key

☐ (B) Stolen Hobinrood's private key

☐ (C) Stolen Hobinrood's certificate

☐ (D) None of the above

☐ (E) ——

☐ (F) ——

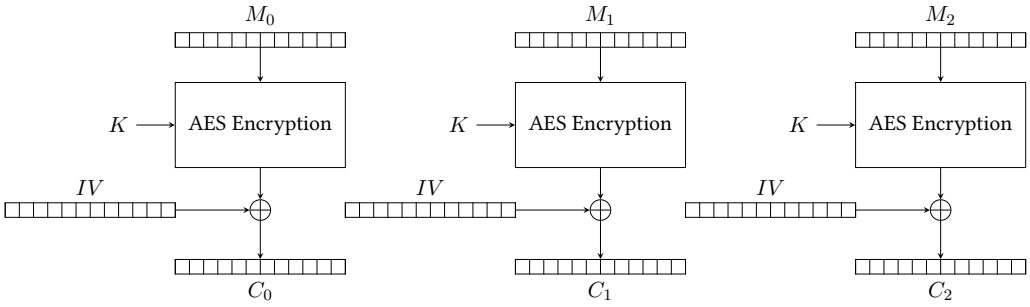
This is the end of Q3. Leave the remaining subparts of Q3 blank on Gradescope, if there are any. Proceed to Q4 on your answer sheet.

Q4 *IV League* **(15 points)**

In this question, E denotes AES block cipher encryption.

Q4.1 (3 points) Recall that AES-ECB is not IND-CPA secure because it is deterministic. What if we tried to introduce randomness to AES-ECB? Consider a new scheme AES-ECB-IV whose construction is as follows:

$$\text{AES-ECB-IV}(K, M) = IV \| C_1 \| \cdots \| C_n$$
$$C_i = E(K, M_i) \oplus IV$$



Note that IV is the same for every block when encrypting a message. Assume IV is randomly generated for each encrypted message. Is AES-ECB-IV IND-CPA secure?

◯ (A) Yes, it is secure even if the attacker can predict future IVs, because it is no longer deterministic.

◯ (B) No, because AES is a bijective (one-to-one) function.

◯ (C) No, because an attacker can still detect when the same block is encrypted twice.

◯ (D) Yes, but only if the attacker is unable to predict future IVs.

◯ (E) ——

◯ (F) ——

For the following parts, consider this new AES scheme below.

$$\text{AES-MULTI}(K, M) = E(K, IV \oplus M_1 \oplus M_2 \oplus \cdots \oplus M_n).$$

AES-MULTI splits the message M into blocks of the appropriate size matching the underlying block cipher. It XORs all of the message blocks together, and then XORs this result with the IV. The result's size is one block, which is fed into the block cipher. The output of the block cipher is the ciphertext.

Q4.2 (3 points) Alice encrypts a message with AES-MULTI. Can Bob decrypt the message?

◯ (G) Yes, but only if the message is one block long.

◯ (H) No, Bob can never decrypt.

◯ (I) Yes, but only if the message is more than one block long.

◯ (J) Yes, Bob can always decrypt.

◯ (K) ——

◯ (L) ——

Q4.3 (3 points) Eve intercepts a ciphertext encrypted with AES-MULTI. Can Eve learn any information about the plaintext?

◯ (A) Yes, but only if the message is one block long.

◯ (B) Yes, Eve can always learn something about the plaintext.

◯ (C) No, Eve can never learn anything about the plaintext.

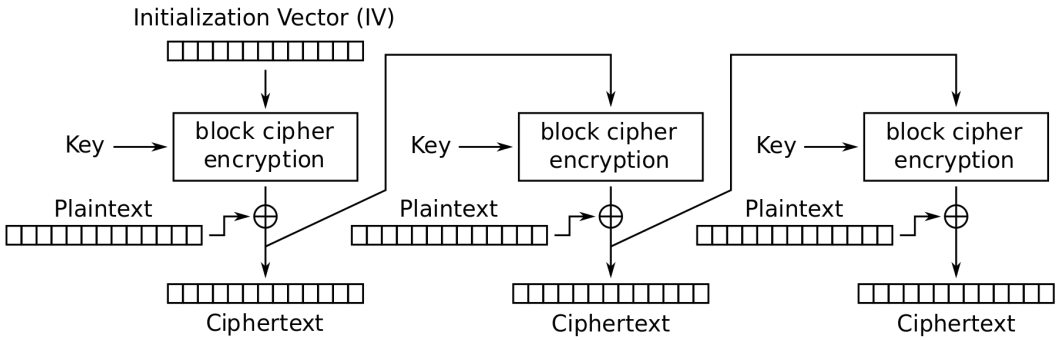
◯ (D) Yes, but only if the message is more than one block long.

◯ (E) ——

◯ (F) ——

The following parts are independent of the previous parts.

Q4.4 (3 points) Recall CFB mode encryption: $C_i = M_i \oplus E(K, C_{i-1}), C_0 = IV$



Cipher Feedback (CFB) mode encryption

Alice and Bob are using AES-CFB with reused IVs. What values can an eavesdropper Eve learn? Select all that apply.

☐ (G) The secret key

☐ (H) Partial information about the plaintexts

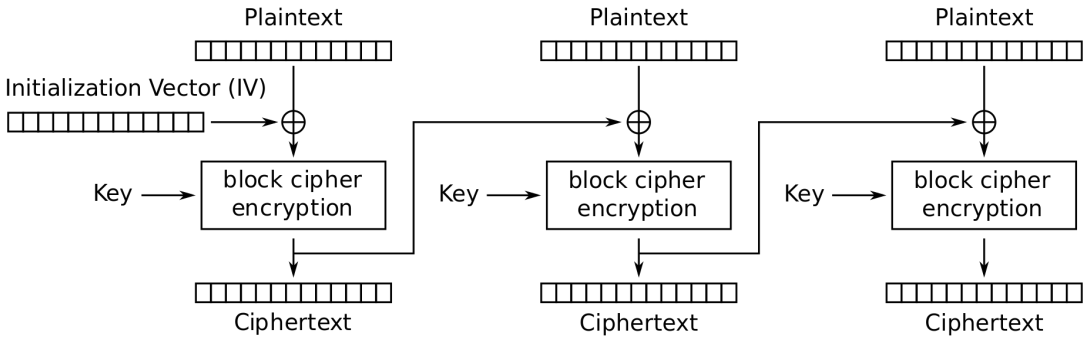
☐ (I) The exact length of the messages

☐ (J) None of the above

☐ (K) ——

☐ (L) ——

Q4.5 (3 points) Recall CBC mode encryption: $C_i = E(K, M_i \oplus C_{i-1}), C_0 = IV$



Cipher Block Chaining (CBC) mode encryption

Alice and Bob are using AES-CBC with reused IVs. Additionally, Alice and Bob prepend a shared counter, incremented per message, to each message before it is encrypted. For example, if Alice's first message is "hello" and Bob's reply is "world", Alice will send "1 - hello" and Bob will send "2 - world" encrypted the same key and IV.

What values can an eavesdropper Eve learn? Select all that apply.

☐ (A) The secret key

☐ (B) Partial information about the plaintexts

☐ (C) The exact length of the messages

☐ (D) None of the above

☐ (E) ——

☐ (F) ——

This is the end of Q4. Leave the remaining subparts of Q4 blank on Gradescope, if there are any. Proceed to Q5 on your answer sheet.

Q5 *Copy Buffers* **(19 points)**

Consider the following vulnerable C code:

```
1  void copy_buffers(char* dst, char* src, size_t num) {
2      strncpy(dst, src, num);
3  }
4
5  int main() {
6      int size_bytes;
7      struct {
8          char x[64];
9          char y[8];
10     } my_struct;
11     char the_buffer[64];
12     size_bytes = sizeof(the_buffer);
13
14     printf("What would you like to write into the_buffer?\n");
15     fgets(the_buffer, size_bytes, stdin);
16
17     copy_buffers(my_struct.y, the_buffer, size_bytes);
18
19     return 0;
20 }
```

Definitions of relevant C functions may be found on the last page of this exam.

Assume you are on a little-endian 32-bit x86 system. Assume that there is no compiler padding or saved registers in all questions.

For this question, assume that **no memory safety defenses** are enabled.

Assume that you have set a breakpoint at line 2 in the program and stopped just before the call to `strncpy`. Fill in the numbered blanks corresponding to the following entries in the stack diagram. Each blank represents a variable or struct member and may represent more than one word. Higher-numbered addresses are located at the top of the diagram.

Stack
RIP of main
SFP of main
(1a)
(1b)
(1c)
(1d)
(2a)
(2b)
(2c)
RIP of copy_buffers
SFP of copy_buffers

Q5.1 (3 points) Section 1:

◯ (A) (1a) = `my_struct.y`; (1b) = `my_struct.x`; (1c) = `the_buffer`; (1d) = `size_bytes`

◯ (B) (1a) = `my_struct.x`; (1b) = `my_struct.y`; (1c) = `the_buffer`; (1d) = `size_bytes`

◯ (C) (1a) = `size_bytes`; (1b) = `my_struct.y`; (1c) = `my_struct.x`; (1d) = `the_buffer`

◯ (D) (1a) = `the_buffer`; (1b) = `my_struct.x`; (1c) = `my_struct.y`; (1d) = `size_bytes`

◯ (E) (1a) = `size_bytes`; (1b) = `my_struct.x`; (1c) = `my_struct.y`; (1d) = `the_buffer`

◯ (F) (1a) = `the_buffer`; (1b) = `my_struct.y`; (1c) = `my_struct.x`; (1d) = `size_bytes`

Q5.2 (3 points) Section 2:

◯ (G) (2a) = `num`; (2b) = `dst`; (2c) = `src`

◯ (H) (2a) = `src`; (2b) = `dst`; (2c) = `num`

◯ (I) (2a) = `dst`; (2b) = `src`; (2c) = `num`

◯ (J) (2a) = `num`; (2b) = `src`; (2c) = `dst`

◯ (K) —

◯ (L) —

Using GDB, you find that the address of the RIP of `main` is `0xfff1f370`. Construct an input that would cause the vulnerable program to execute shellcode when provided to the program.

Q5.3 (5 points) The first part of your input should be some number of garbage bytes. How many bytes of garbage do you need? Your answer should be an integer. *Enter your answer in the text box on Gradescope.*

◯ (A) — ◯ (B) — ◯ (C) — ◯ (D) — ◯ (E) — ◯ (F) —

Q5.4 (5 points) The remainder of your input should be a series of bytes. What should these bytes be? You may use the variable `SHELLCODE` as 30-byte shellcode byte sequence. Your answer should be an expression in Python 2 syntax (just like Project 1). *Enter your answer in the text box on Gradescope.*

◯ (G) — ◯ (H) — ◯ (I) — ◯ (J) — ◯ (K) — ◯ (L) —

Q5.5 (3 points) Which of the following defenses would individually stop your exploit from the previous parts? Select all that apply.

☐ (A) ASLR

☐ (B) Non-executable pages (also called DEP, W^X, and the NX bit)

☐ (C) Stack canaries

☐ (D) None of the above

☐ (E) ——

☐ (F) ——

This is the end of Q5. Leave the remaining subparts of Q5 blank on Gradescope, if there are any. Proceed to Q6 on your answer sheet.

Q6 *Socially Distant Coin Flipping* (18 points)

Alice and Bob want to flip a coin to settle a bet, but they can't meet in person. They both suspect that the other person might try to cheat to win the bet, so they need your help to construct a cryptographic coin-flipping scheme.

In general, a coin-flipping scheme works as follows:

1. Alice makes a guess b (where b is a bit, 0 for heads and 1 for tails). She locks in her guess by generating a value $C(b)$ called the *commitment*. Alice sends the commitment to Bob.
2. Bob flips the coin and reports the result of the flip to Alice.
3. Alice reveals her guess b and optionally sends some additional information for verification. Bob can use this additional information to check that Alice's revealed guess matches her commitment.

A secure coin-flipping scheme must have two properties to prevent cheating:

- **Hiding**: The commitment $C(b)$ without any additional verification information must leak no information about Alice's guess b. Otherwise, Bob could cheat by deducing that Alice guessed heads and claim that he flipped tails.
- **Binding**: The commitment must bind Alice to her guess—that is, Alice should not be able to change her guess after she has sent her commitment to Bob. In other words, Alice should not be able to guess heads, send a commitment for heads, and then claim that she guessed tails, without being detected by Bob.

For each scheme below, determine whether a scheme fulfills the binding property, the hiding property, both, or neither. In all questions, $\|$ denotes concatenation.

Q6.1 (3 points) Commitment: Alice sends her guess to Bob: $C(b) = b$.

Verification: Alice reveals her guess. Bob checks that Alice's guess matches her commitment, i.e. he checks that $C(b) = b$.

◯ (A) Binding only ◯ (C) Neither property ◯ (E) ——

◯ (B) Hiding only ◯ (D) Both properties ◯ (F) ——

Q6.2 (6 points) In this part, p is a publicly known, large prime number; g is a publicly known generator modulo p; and a is another publicly known large number modulo p.

Commitment: Alice calculates $C(b) = g^{a+b} \bmod p$.

Verification: Alice reveals her guess. Bob checks that $C(b) = g^{a+b} \bmod p$.

If you answered that the scheme is hiding, write one sentence explaining why. If you answered that the scheme is not hiding, write one sentence explaining how Bob can learn Alice's guess.

Enter your answer in the text box on Gradescope.

◯ (G) Binding only ◯ (I) Neither property ◯ (K) ——

◯ (H) Hiding only ◯ (J) Both properties ◯ (L) ——

Q6.3 (6 points) Commitment: Alice picks a random bit b' and XORs her guess with that bit: $C(b) = b \oplus b'$.

Verification: Alice reveals her guess and the random bit b'. Bob checks if Alice's guess, XORed with b', is equal to her commitment, i.e. he checks that $C(b) = b \oplus b'$.

If you answered that the scheme is binding, write one sentence explaining why. If you answered that the scheme is not binding, write one sentence explaining how Alice can guess heads (0) and claim that she guessed tails (1).

Enter your answer in the text box on Gradescope.

◯ (A) Binding only ◯ (C) Neither property ◯ (E) ——

◯ (B) Hiding only ◯ (D) Both properties ◯ (F) ——

Q6.4 (3 points) Commitment: Alice generates a random secret key k and then encrypts her guess with a secure block cipher: $C(b) = E(k, b)$.

Verification: Alice reveals her guess and the key k. Bob decrypts the commitment and verifies that it matches Alice's guess, i.e. he checks that $D(k, C(b)) = b$.

Assume that encryption will automatically pad to the block size and decryption will unpad to the orginal message.

◯ (G) Binding only ◯ (I) Neither property ◯ (K) ——

◯ (H) Hiding only ◯ (J) Both properties ◯ (L) ——

This is the end of Q6. Leave the remaining subparts of Q6 blank on Gradescope, if there are any. Proceed to Q7 on your answer sheet.

Q7 *Palindromify* (31 points)

Consider the following C code:

```
1  struct flags {
2      char debug[4];
3      char done[4];
4  };
5
6  void palindromify(char *input, struct flags *f) {
7      size_t i = 0;
8      size_t j = strlen(input);
9
10     while (j > i) {
11         if (input[i] != input[j]) {
12             input[j] = input[i];
13             if (strncmp("BBBB", f->debug, 4) == 0) {
14                 printf("Next: %s\n", input);
15             }
16         }
17         i++; j--;
18     }
19 }
20
21 int main(void) {
22     struct flags f;
23     char buffer[8];
24     while (strncmp("XXXX", f.done, 4) != 0) {
25         gets(buffer);
26         palindromify(buffer, &f);
27     }
28     return 0;
29 }
```

Definitions of relevant C functions may be found on the last page of this exam.

Assume you are on a little-endian 32-bit x86 system. Assume that there is no compiler padding or saved registers in all questions.

For parts 1–3, assume that **no memory safety defenses** are enabled.

Q7.1 (3 points) Which of the following lines contains a memory safety vulnerability?

- ◯ (A) Line 24
- ◯ (B) Line 12
- ◯ (C) Line 25
- ◯ (D) Line 10
- ◯ (E) —
- ◯ (F) —

Q7.2 (3 points) Which of these inputs would cause the program to execute shellcode located at `0xbffd35a0`?

◯ (G) `(20 * 'X') + '\xa0\x35\xfd\xbf'`

◯ (H) `'\x00' + (11 * 'A') + (4 * 'X') + (4 * 'A') + '\xa0\x35\xfd\xbf'`

◯ (I) `'\x00' + (7 * 'A') + (4 * 'X') + (4 * 'A') + '\xa0\x35\xfd\xbf'`

◯ (J) `'\x00' + (19 * 'A') + '\xa0\x35\xfd\xbf'`

◯ (K) `(16 * 'X') + '\xa0\x35\xfd\xbf'`

◯ (L) None of the above

Q7.3 (3 points) Assume you did the previous part correctly. At what point will the instruction pointer jump to the shellcode?

◯ (A) Immediately after `gets` returns

◯ (B) Immediately after `main` returns

◯ (C) Immediately after `palindromify` returns

◯ (D) Immediately after `printf` returns

◯ (E) ——

◯ (F) ——

For parts 4–7, assume that stack canaries are enabled, and **all 4 bytes of the canary are random and not null**. Assume that `gets` will append a single null byte to your input.

Q7.4 (5 points) Which of the following values on the stack can we overwrite without writing to the stack canary? Select all that apply.

☐ (G) RIP of `main`

☐ (H) SFP of `main`

☐ (I) `f`

☐ (J) RIP of `palindromify`

☐ (K) `buffer`

☐ (L) None of the above

Q7.5 (3 points) Suppose that we provide `ABCDE` as input to the program. When we enter the `palindromify` function, what will be the initial value of `j`?

◯ (A) 0 ◯ (B) 1 ◯ (C) 2 ◯ (D) 3 ◯ (E) 4 ◯ (F) 5

Q7.6 (5 points) Provide the **first** line of an input that will allow you to redirect execution of this program to shellcode located at `0xbffd35a0`. Write your answer in Python 2 syntax (just like Project 1). *Enter your answer in the text box on Gradescope.*

◯ (G) —— ◯ (H) —— ◯ (I) —— ◯ (J) —— ◯ (K) —— ◯ (L) ——

Q7.7 (5 points) Provide the **second** line of an input that will allow you to redirect execution of this program to shellcode located at `0xbffd35a0`. You can use `out` as a variable that contains the output from the first input. Write your answer in Python 2 syntax (just like Project 1). *Enter your answer in the text box on Gradescope.*

◯ (A) —— ◯ (B) —— ◯ (C) —— ◯ (D) —— ◯ (E) —— ◯ (F) ——

Q7.8 (4 points) Assume the shellcode from the earlier parts resides in the stack section of memory. Which of the following would we be able to do if stack canaries and ASLR were both in use? Select all that apply.

☐ (G) Leak the stack canary

☐ (H) Overwrite the value of `struct flags f`

☐ (I) Redirect execution to the shellcode using the method from parts 6–7

☐ (J) Overwrite the value of `i` and `j`

☐ (K) None of the above

☐ (L) ——

This is the end of Q7. Leave the remaining subparts of Q7 blank on Gradescope, if there are any. You have reached the end of the exam.

C Function Definitions

```
size_t strlen(const char *s);
```

The strlen() function calculates the length of the string pointed to by s, excluding the terminating null byte ('\0').

```
int strncmp(const char *s1, const char *s2, size_t n);
```

The strncmp() function compares the first (at most) n bytes of two strings s1 and s2. It returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.

```
char *strncpy(char *dest, const char *src, size_t n);
```

The strncpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and at most n bytes of s are copied. Warning: If there is no null byte among the first n bytes of src, the string placed in dest will not be null-terminated.

If the length of src is less than n, strncpy() writes additional null bytes to dest to ensure that a total of n bytes are written.

```
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').

```
char *fgets(char *s, int size, FILE *stream);
```

fgets() reads in at most one less than size characters from stream and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer