Q1 Fizzbuzz

13 Points

Homework assignments have instant feedback enabled. When you click "Save Answer," if the answer is correct, you will see an explanation.

You can resubmit as many times as you want until the due date. After the due date, to avoid being marked late, do not submit again.

This question shows you how to defeat stack canaries to exploit a program. We recommend trying this question before doing Project 1, Question 3.

You just finished implementing interactive fizzbuzz with a custom error message, shown below:

```
void fizzbuzz(int *return_code, char *error_msg,
              char *input) {
    int x = atoi(input);
    // C atoi returns an int if string can be converted
    // or 0 otherwise
    if (x == 0) {
        *return_code = 0xBADCA75;
        //it just has to be nonzero, right?
        printf("%s", error_msg);
    } else {
        if (x % 3 == 0){
            printf("fizz");
        }
        if (x % 5 == 0){
            printf("buzz");
        }
        if (x % 3 != 0 && x % 5 != 0){
            printf("%d", x);
        }
    }
    printf("\n");
}

int main() {
    int return_code = 0;
    char error_msg[100];
    char input[20];
    gets(error_msg);
    while (has_input()) {
        gets(input);
        fizzbuzz(&return_code, error_msg, input);
    }
    return return_code;
}
```

Assume that this code runs with a completely random 32-bit stack canary. The stack canary is a value placed between the saved ebp and local variables. If the value of stack canary changes, the code will crash before returning and prevent any malicious code from being executed. This is potentially useful as an overflow from a local buffer will overwrite the canary before overwriting the saved eip.

Assume no other memory safety defenses, no exception handlers, no callee saved registers, no compiler optimizations, and that local variables are stored in the stack in order as they appear in the code (for example in the `main` frame `return_code` will be at a higher memory address than `input`).

EvanBot believes that this code is vulnerable to a buffer overflow attack, even with the stack canary enabled.

Q1.1
1 Point

EvanBot suggests first drawing a stack diagram. Remember to include the stack canary in your diagram.

According to your stack diagram, which of the following are vulnerable to being overwritten by user input as a result of the `gets` calls, without causing the program to crash?

- □ int x
- ☑ int return_code
- ☑ char error_msg[100]
- ☑ char input[20]
- □ rip of main
- □ rip of fizzbuzz
- □ None of the above

Q1.2
1 Point

EvanBot suggests finding the value of the stack canary for the `main` stack frame before `main` returns.

If we know the value of the stack canary, which of the following are vulnerable to being overwritten by user input as a result of the `gets` calls, without causing the program to crash?

- □ int x
- ☑ int return_code
- ☑ char error_msg[100]
- ☑ char input[20]
- ☑ rip of main
- □ rip of fizzbuzz
- □ None of the above

Q1.3
1 Point

Recall that strings in C are null-terminated. When writing user input to memory, `gets` automatically appends a null byte to the end of the string. When printing out a string, `printf` dereferences a pointer to the string (passed as an argument to `printf`) and prints until it encounters a null byte.

Given this information, which of the following lines of code will cause the canary to leak?

Hint: To leak the canary, the code must produce some sort of output.

○ `int x = atoi(input);`

● `printf("%s", error_msg);`

○ `printf("%d", x)`

○ `gets(error_msg);`

○ None of the above

Q1.4
1 Point

Provide an initial `error_msg` and the first `input` that will reveal the stack canary when the program is run.

Your goal is to remove all null bytes between the place where `printf` starts printing and the stack canary. Remember that `gets()` automatically appends a null byte to the end of the string.

`error_msg = <'A' repeated ___ times>`

The first blank:

100

`input = '___'`, where `input` MUST be a number

Hint: What `input` causes the vulnerable line of code to run?

0

Q1.5
1 Point

What is the probability that this exploit works? The entire canary value must be printed for the exploit to work.

Hint: Recall that the stack canary is four completely random bytes.

○ $\left(\frac{1}{2^8}\right)^4$

● $\left(1 - \frac{1}{2^8}\right)^4$

○ $1 - \left(\frac{1}{2^8}\right)^4$

○ $\left(\frac{1}{2^8 - 1}\right)^4$

Q1.6
1 Point

Your code would not have been subject to the specific vulnerability above if you used a different return code on error. Which of the following return codes would have prevented the exploit?

□ 0x900DBEEF

□ 0xD00DFACE

□ 0xFFFFFFFF

☑ 0xD00B1D00

Q1.7
1 Point

Give the second `input` that will cause a shell to spawn. You can assume the `main` function returns after this input. You have access to following values:

- `SHELLCODE`, 65-byte set of instructions that spawns a shell.
- `CANARY`, 4-byte value you found in the previous part
- `ESP_MAIN`, 4-byte bottom (lowest) address of `main` stack frame, after all local variables are initialized
- `ESP_FIZZBUZZ`, 4-byte bottom (lowest) address of `fizzbuzz` stack frame, after all local variables are initialized

If the value of the bytes don't matter for your input, please select garbage instead of using one of the placeholders. (For example, if you need 65 bytes of garbage, select "Bytes of garbage" and type `65` in the box instead of selecting `SHELLCODE`.)

Hint: It might help to refer back to the stack diagram for these parts.

Note: If you choose one of the first four options, you may need to put a space in the box for Gradescope to mark your answer correct.

First, input:

● SHELLCODE
○ CANARY
○ ESP_MAIN
○ ESP_FIZZBUZZ
○ Bytes of garbage (specify how many in the box below)

Q1.8
1 Point

Then input:

○ SHELLCODE

○ CANARY

○ ESP_MAIN

○ ESP_FIZZBUZZ

● Bytes of garbage (specify how many in the box below)

59

Q1.9
1 Point

Then input:

○ SHELLCODE

● CANARY

○ ESP_MAIN

○ ESP_FIZZBUZZ

○ Bytes of garbage (specify how many in the box below)

Q1.10
1 Point

Then input:

○ SHELLCODE

○ CANARY

○ ESP_MAIN

○ ESP_FIZZBUZZ

● Bytes of garbage (specify how many in the box below)

4

Q1.11
1 Point

Then input:

○ SHELLCODE

○ CANARY

● ESP_MAIN

○ ESP_FIZZBUZZ

○ Bytes of garbage (specify how many in the box below)

Q1.12
1 Point

Would your exploit still work if the `while` loop in main was removed, so that multiple inputs required multiple executions of the program?

○ Yes, with no modifications

○ Yes, with minor modifications

● No

Q1.13
1 Point

Would your exploit still work if non-executable pages (also known as DEP, W^X, or the NX bit) were enabled?

○ Yes, with no modifications

○ Yes, with minor modifications

● No

Q2 Printf Oracle

7 Points

Consider the following C snippet:

```
void oracle() {
   char string[80];
   fgets(string, 80, stdin);
   printf(string);
}

int main() {
   oracle();
   return 0;
}
```

For all parts of this question, `input` is the standard input given by the user when `fgets` is called. Assume there are no exception handlers, no callee saved registers, and no variables are optimized out.

No buffer overflow protection is enabled. This program is run on a 32-bit x86 machine.

Q2.1

1 Point

Which line of code is vulnerable?

○ `fgets(string, 80, stdin);`

● `printf(string);`

How should this line of code be fixed?

○ `fgets(string, sizeof(string), stdin);`

○ `gets(string);`

● `printf("%s", string);`

○ `printf("%d", string);`

Q2.2
1 Point

True or false: stack canaries will prevent an attacker from exploiting this vulnerability.

○ True

● False

Q2.3
1 Point

Which inputs will cause the program to crash, with high probability?

Hint: Which inputs involve dereferencing a pointer?

☐ %c

☐ %d

☑ %n

☑ %s

☐ %u

☐ %x

Q2.4
1 Point

Which inputs will cause the program to leak values from the stack?

Hint: the inputs you chose in the previous part cause the program to crash, so those won't cause the program to leak values from the stack.

- ☑ %c
- ☑ %d
- □ %n
- □ %s
- ☑ %u
- ☑ %x

Q2.5
1 Point

Which format type should an attacker use to read memory from addresses **outside the stack**?

- ○ %c
- ○ %d
- ○ %n
- ● %s
- ○ %u
- ○ %x

Q2.6
1 Point

Which format type should an attacker use to **write** to memory at addresses outside the stack?

○ %c

○ %d

● %n

○ %s

○ %u

○ %x

Q2.7
1 Point

Which of these inputs would cause `oracle` to print 100 characters while still fitting in the 80 characters provided for input?

Hint: Take a look at `man 3 printf`, or [the Wikipedia page on `printf` format strings](#).

□ 'a' * 100

☑ %100c

□ None of the above

Q3 AES-ENC

6 Points

EvanBot decides to create a new block cipher mode, called AES-ENC (EvanBot Novel Cipher). It is defined as follows:

$$C_i = E_k(C_{i-1}) \oplus P_i$$

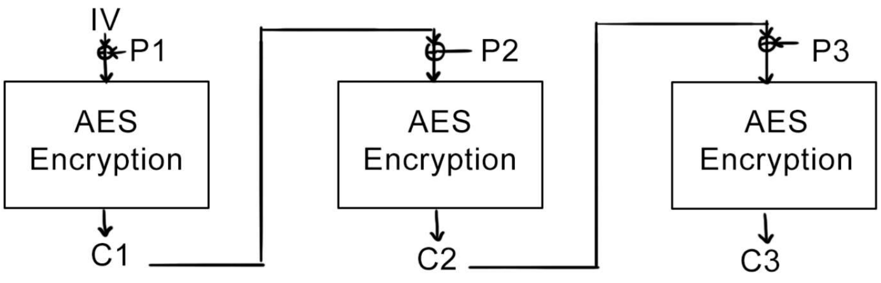
$$C_0 = \mathrm{IV}$$

$(P_1, \ldots, P_n)$ are the plaintext messages, E_K is block cipher encryption with key K.

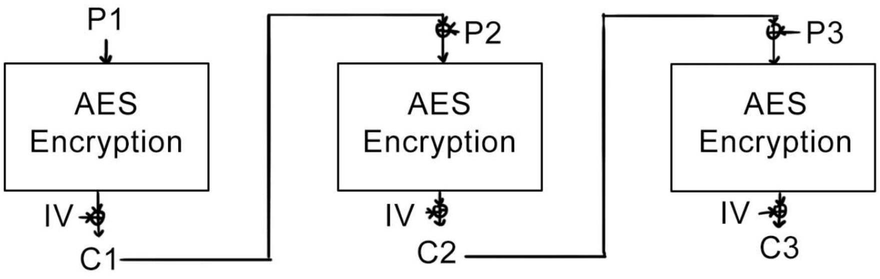
Q3.1
1 Point

Select the correct encryption diagram for AES-ENC.

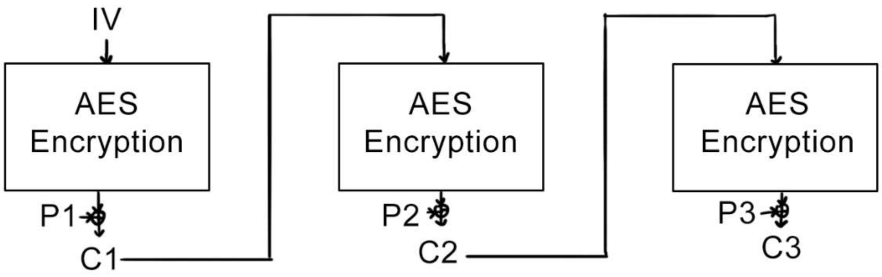
○



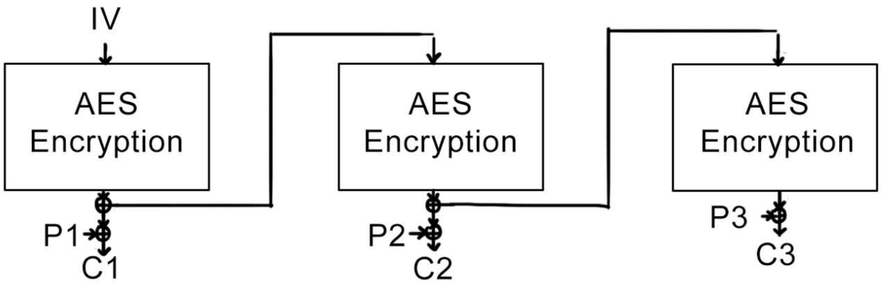
○



●



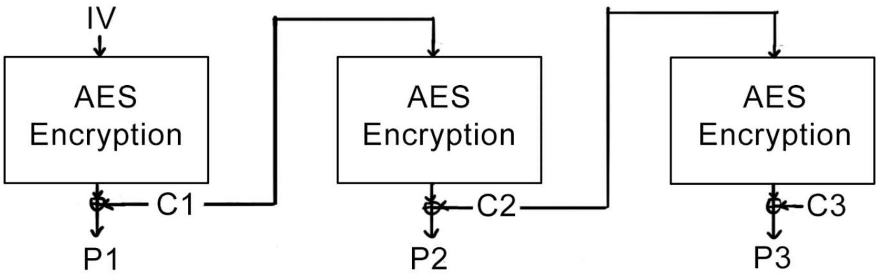
○



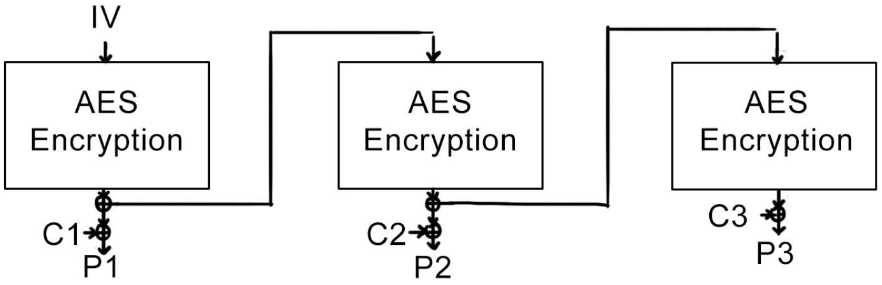
Q3.2
1 Point

Select the correct decryption diagram for AES-ENC.

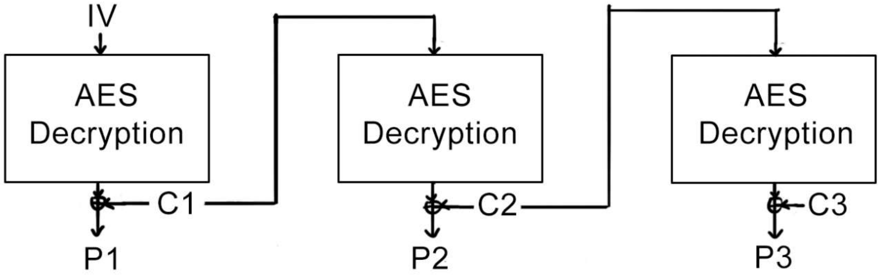
●



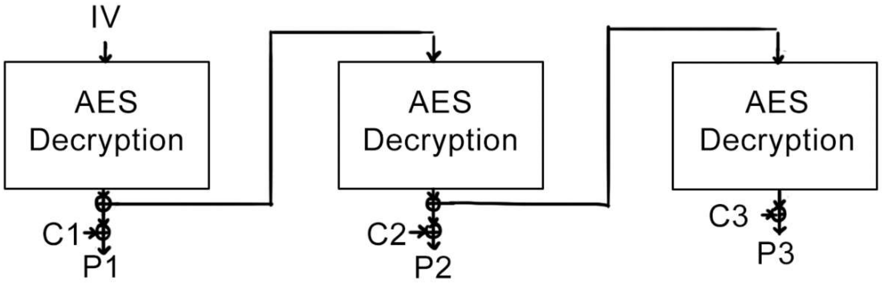
○



○



○



Q3.3
1 Point

Is AES-ENC encryption parallelizable?

○ Yes

● No

How about decryption?

● Yes

○ No

Q3.4
1 Point

As we saw (or will see) in discussion, AES-CBC is vulnerable to a chosen plaintext attack when the IV which will be used to encrypt the message is known in advance. Is AES-ENC vulnerable to the same issue?

○ Yes, because a specially crafted input can "cancel out" the IV.

○ Yes, but not for the reason above.

● No, because the IV passes through the AES encryption block.

○ No, but not for the reason above.

Q3.5
1 Point

Suppose that Alice means to send the message $(P_1, \ldots, P_n)$ to Bob using AES-ENC. By accident, Alice typos and encrypts $(P_1, P_2 \oplus 1, P_3, \ldots, P_n)$ instead (i.e., she accidentally flips the last bit of the second block).

Select the ciphertext block(s) that will **NOT** decrypt to correct plaintext.

□ First block

☑ Second block

□ Third block

□ Subsequent blocks

Q3.6
1 Point

Alice encrypts the message $(P_1, \ldots, P_5)$. Unfortunately, the block C_2 of the ciphertext is lost in transmission, so that Bob receives $(C_0, C_1, C_3, C_4, C_5)$. Assuming that Bob knows that he is missing the second ciphertext block C_2, which blocks of the original plaintext can Bob recover?

☑ P_1

□ P_2

□ P_3

☑ P_4

☑ P_5

Q4 Padding

5 Points

Recall that block ciphers can only encrypt messages of a fixed size, which is called the *block size*. We know that we can use block chaining modes (e.g. CBC mode) to deal with messages that are longer than the block size, but they don't solve the problem of messages whose lengths aren't an integer multiple of the block size. So how do we make do? We add *padding*. For this question, we'll assume that the block cipher we're using is AES, which uses 16-byte blocks.

Q4.1

1 Point

Consider a padding scheme that adds `0`s to the end of the message until its length is a multiple of 16. For example, the message `PANCAKES`:

```
0 1 2 3 4 5 6 7
P A N C A K E S
```

would be padded to become:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
P A N C A K E S 0 0 0  0  0  0  0  0
```

Can this padding scheme correctly pad and de-pad messages? (In other words, if you pad a message and then de-pad it, will you get the original message back?)

○ Yes, for all messages

● Yes, but not for all messages

○ No

Q4.2
1 Point

Consider a padding scheme that takes the last byte of a message and repeatedly appends copies of that byte until the message length is a multiple of 16. For example, the message `PANCAKES`:

```
0 1 2 3 4 5 6 7
P A N C A K E S
```

would be padded to become:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
P A N C A K E S S S S  S  S  S  S  S
```

Can this padding scheme correctly pad and de-pad messages? (In other words, if you pad a message and then de-pad it, will you get the original message back?)

○ Yes, for all messages

● Yes, but not for all messages

○ No

Q4.3
1 Point

Consider another padding scheme: instead of padding with all `0`s, the value of our pad is the number of bytes of padding that we added. Since we need to add 8 bytes of padding, our message `PANCAKES` would be padded to become:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
P A N C A K E S 8 8 8  8  8  8  8  8
```

Note that in if the message is a multiple of the block size, another block with 16 `16`s is added.

Can this padding scheme correctly pad and de-pad messages? (In other words, if you pad a message and then de-pad it, will you get the original message back?)

● Yes, for all messages

○ Yes, but not for all messages

○ No

Q4.4
1 Point

Which of the following 16-byte messages have valid PKCS#7 (the scheme from the previous part) padding? In other words, which messages could we correctly de-pad?

☑ EVANBOT999999999

□ EVANBOT123456789

☑ EVANBOT987654321

☑ EVANBOT987654322

Q4.5

1 Point

Suppose you are an attacker, and you intercept an unencrypted, padded plaintext message.

Can you change the last byte to a constant value that guarantees that the message has valid padding?

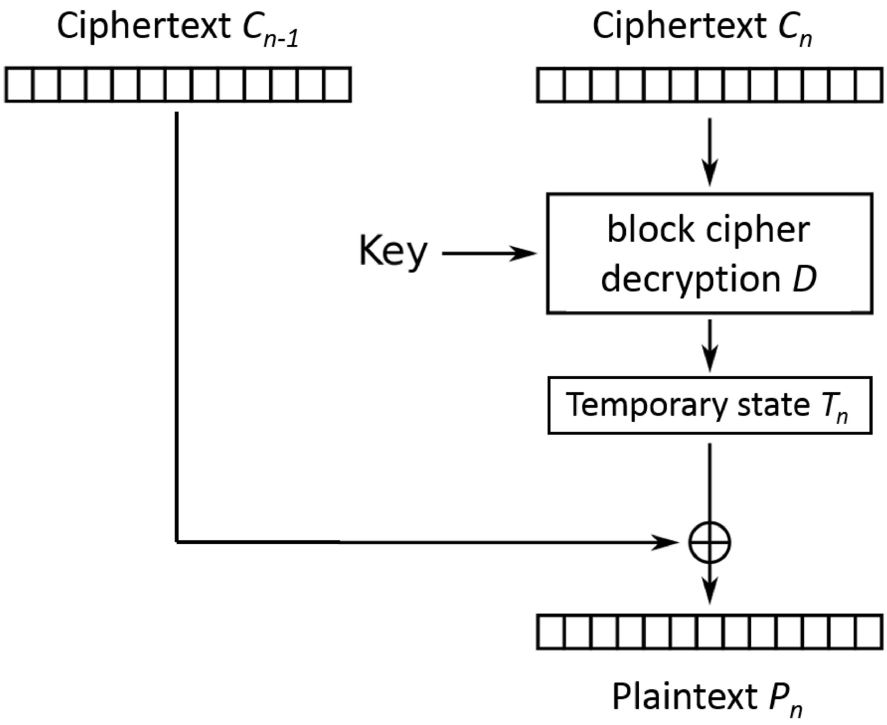
If yes, enter the constant value below (a number between 0 and 16). If no, type "No" below.

1

Q5 CBC Review

3 Points

Recall decryption in CBC mode. In particular, we are interested in the decryption of a single block of plaintext — especially in the temporary block state that occurs before the XOR:



Q5.1

1 Point

Which of these is a correct expression for T_n?

● $D(C_n)$

○ C_n

○ C_{n-1}

Q5.2
1 Point

Which of these is a correct expression for P_n?

☑ $C_{n-1} \oplus D(C_n)$

☑ $C_{n-1} \oplus T_n$

□ $D(C_n) \oplus C_n$

□ $D(C_n) \oplus D(C_{n-1})$

Q5.3
1 Point

At least one of your potential answers to the previous part should give you an equation that relates P_n, C_{n-1}, and T_n. Solve this equation and find an expression for T_n in terms of P_n and C_{n-1}.

○ $T_n = C_n \oplus C_{n-1}$

● $T_n = P_n \oplus C_{n-1}$

○ $T_n = P_n \oplus C_n$

Q6 Padding Oracles

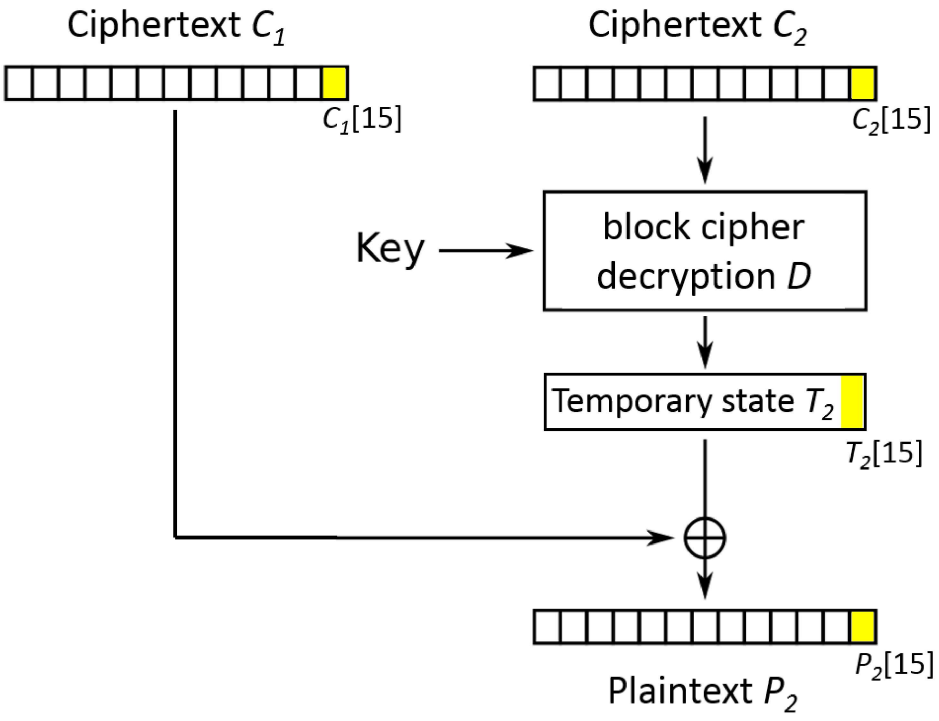
4 Points

Next, let's introduce the concept of a *padding oracle*.

A padding oracle is a black-box function which takes as its input some ciphertext `c`, and returns `True` if the (decrypted) ciphertext is properly padded and `False` otherwise. Note that the padding oracle has access to the secret key k used for encryption and decryption.

Assume you've intercepted a two-block ciphertext (IV, C_1, C_2), and you have access to a padding oracle. This means you can send the oracle *arbitrary* inputs, and it will decrypt your input using k and truthfully report whether it is padded correctly.

For your convenience, here is the CBC diagram for a two-block message.



Q6.1
1 Point

Our goal for this question is to modify the ciphertext so that its plaintext decryption has valid padding no matter what. One way to do this is to modify the last byte of C_1, which we will denote as $C_1[15]$.

Which modified value of $C_1[15]$ will cause the padding oracle to always report that the decryption has valid padding?

Hint: Use Q4.5 and Q5.2.

○ $C_1[15]$

○ $T_1[15]$

○ $C_2[15]$

○ $T_2[15]$

○ $C_1[15] \oplus 1$

○ $T_1[15] \oplus 1$

○ $C_2[15] \oplus 1$

● $T_2[15] \oplus 1$

Q6.2
1 Point

Let $C_1'[15]$ denote the modified ciphertext byte in the previous part that always results in correct padding.

Which of these expressions evaluates to the value of $P_2[15]$, the last byte of P_2?

Hint: Start with your solution to the previous part, and use Q5.2 to relate $P_2[15]$ to the equation you found in the previous part.

● $C_1[15] \oplus C_1'[15] \oplus 1$

○ $C_1[15] \oplus C_2'[15] \oplus 1$

○ $C_2[15] \oplus C_1'[15] \oplus 1$

○ $C_2[15] \oplus C_2'[15] \oplus 1$

○ $C_1[15] \oplus C_1'[15]$

○ $C_1[15] \oplus C_2'[15]$

○ $C_2[15] \oplus C_1'[15]$

○ $C_2[15] \oplus C_2'[15]$

Q6.3
1 Point

Now let's modify the attack above to learn $P_2[14]$, the second-to-last byte of P_2. To do this, we'll modify $C_1[14]$ and $C_1[15]$, the last two bytes of C_1.

Which modified value of $C_1[14]$ will cause the padding oracle to always report that the decryption has valid padding?

○ $T_1[14] \oplus 1$

○ $T_2[14] \oplus 1$

○ $T_1[14] \oplus 2$

● $T_2[14] \oplus 2$

Which modified value of $C_1[15]$ will cause the padding oracle to always report that the decryption has valid padding?

○ $T_1[15] \oplus 1$

○ $T_2[15] \oplus 1$

○ $T_1[15] \oplus 2$

● $T_2[15] \oplus 2$

Hint: The "something to think about later" part of the Q4.5 solution.

Q6.4
1 Point

Let $C_1'[14]$ and $C_1'[15]$ denote the modified ciphertext bytes in the previous part that always results in correct padding.

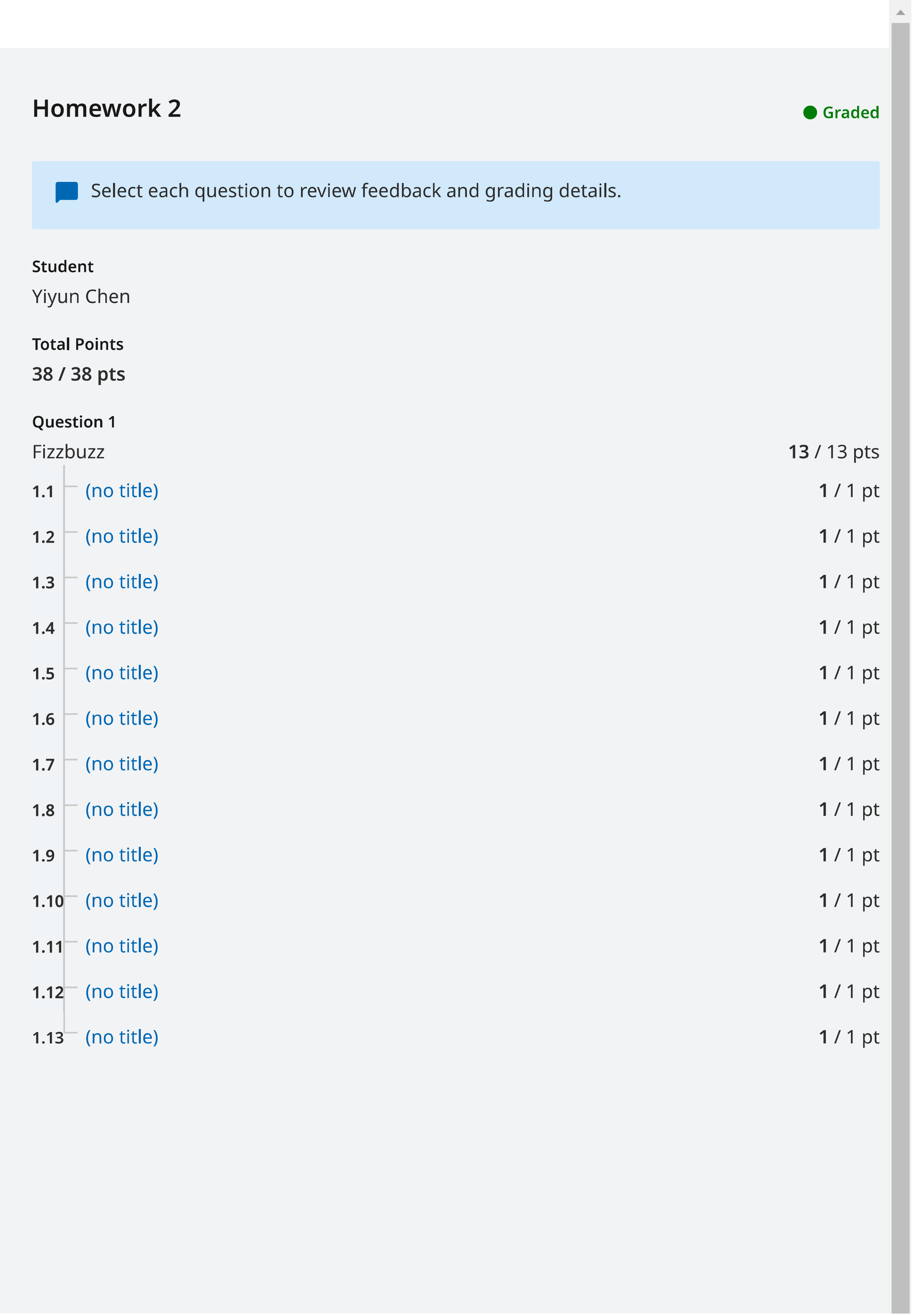
Which of these expressions evaluates to the value of $P_2[14]$, the second-to-last byte of P_2?

○ $C_1[14] \oplus C_1'[14] \oplus 1$
○ $C_1[14] \oplus C_1'[15] \oplus 1$
○ $C_1[15] \oplus C_1'[14] \oplus 1$
○ $C_1[15] \oplus C_1'[15] \oplus 1$
● $C_1[14] \oplus C_1'[14] \oplus 2$
○ $C_1[14] \oplus C_1'[15] \oplus 2$
○ $C_1[15] \oplus C_1'[14] \oplus 2$
○ $C_1[15] \oplus C_1'[15] \oplus 2$

Q7 Feedback
0 Points

Optionally, feel free to include feedback. What's something 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 here.

Your name will not be connected to any feedback you provide, and anything you submit here will not affect your grade.



Homework 2

● Graded

Select each question to review feedback and grading details.

Student

Yiyun Chen

Total Points

38 / 38 pts

Question 1

	Fizzbuzz	**13** / 13 pts
1.1	(no title)	**1** / 1 pt
1.2	(no title)	**1** / 1 pt
1.3	(no title)	**1** / 1 pt
1.4	(no title)	**1** / 1 pt
1.5	(no title)	**1** / 1 pt
1.6	(no title)	**1** / 1 pt
1.7	(no title)	**1** / 1 pt
1.8	(no title)	**1** / 1 pt
1.9	(no title)	**1** / 1 pt
1.10	(no title)	**1** / 1 pt
1.11	(no title)	**1** / 1 pt
1.12	(no title)	**1** / 1 pt
1.13	(no title)	**1** / 1 pt

Question 2

Printf Oracle		**7** / 7 pts
2.1	(no title)	**1** / 1 pt
2.2	(no title)	**1** / 1 pt
2.3	(no title)	**1** / 1 pt
2.4	(no title)	**1** / 1 pt
2.5	(no title)	**1** / 1 pt
2.6	(no title)	**1** / 1 pt
2.7	(no title)	**1** / 1 pt

Question 3

AES-ENC		**6** / 6 pts
3.1	(no title)	**1** / 1 pt
3.2	(no title)	**1** / 1 pt
3.3	(no title)	**1** / 1 pt
3.4	(no title)	**1** / 1 pt
3.5	(no title)	**1** / 1 pt
3.6	(no title)	**1** / 1 pt

Question 4

Padding		**5** / 5 pts
4.1	(no title)	**1** / 1 pt
4.2	(no title)	**1** / 1 pt
4.3	(no title)	**1** / 1 pt
4.4	(no title)	**1** / 1 pt
4.5	(no title)	**1** / 1 pt

Question 5

CBC Review		**3** / 3 pts
5.1	(no title)	**1** / 1 pt
5.2	(no title)	**1** / 1 pt
5.3	(no title)	**1** / 1 pt

Question 6

Padding Oracles		**4** / 4 pts
6.1	(no title)	**1** / 1 pt
6.2	(no title)	**1** / 1 pt
6.3	(no title)	**1** / 1 pt
6.4	(no title)	**1** / 1 pt

Question 7

Feedback	**0** / 0 pts