**Q1** ***Robin*** **(20 points)**

Consider the follov

```
1  void robin(vo
2      char buf[
3      int i;
4
5      if (fread(&i, sizeof(int), 1, stdin) != 1)
6          return;
7
8      if (fgets(buf, sizeof(buf), stdin) == NULL)
9          return;
10
11     _________
12 }
```



Assume that:

- There is no compiler padding or additional saved registers.
- The provided line of code in each subpart compiles and runs.
- `buf` is located at memory address `0x21ffd8d8`.
- Stack canaries are enabled, and all other memory safety defenses are disabled.
- The stack canary is four completely random bytes (**no null byte**).

For each subpart, mark whether it is possible to leak the value of the stack canary. If you put possible, provide an input to Line 5 and an input to Line 8 that would leak the canary. If the line is not needed for the exploit, you must write "Not needed" in the box.

Write your answer in Python syntax.

Q1.1 (3 points) Line 11 contains `gets(buf);`.

○ Possible

○ Not possible

Line 5:

Line 8:

Q1.2 (5 points) **For this subpart only, enter an input that allows you to leak a single character from memory address `0xffffd8d7`. Mark "Not possible" if this is not possible.** Line 11 contains `printf("%c", buf[i]);`.

○ Possible

○ Not possible

Line 5:

Line 8:

Q1.3 (6 points) Line 11 contains `printf(buf);`.

○ Possible

○ Not possible

Line 5:

Line 8:

Q1.4 (6 points) Line 11 contains `printf("%d", i);`.



○ Possible

○ Not possible

Line 5:

Line 8:

**Q2** ***The Way You Look Tonight*** **(20 points)**

Consider the following vulnerable C code:

```
1  typedef struct {
2      char mon[16];
3      char chan
4  } duo;
5
6  void third_wh                    , FILE *f) {
7      duo mond
8      duo richa
9      fgets(ric                    );
10     strcpy(richard.chan, puppet);
11     int8_t alias = 0;
12     size_t counter = 0;
13
14     while (!richard.mon[15] && richard.mon[0]) {
15         size_t index = counter / 10;
16         if (mondler.mon[index] == 'A') {
17             mondler.mon[index] = 0;
18         }
19         alias++;
20         counter++;
21         if (counter == ___ || counter == ___) {
22             richard.chan[alias] = mondler.mon[alias];
23         }
24     }
25
26     printf("%s\n", richard.mon);
27     fflush(stdout); // no memory safety vulnerabilities on this line
28 }
29
30 void valentine(char *tape[2], FILE *f) {
31     int song = 0;
32     while (song < 2) {
33         read_input(tape[song]); //memory-safe function, see below
34         third_wheel(tape[song], f);
35         song++;
36     }
37 }
```

For all of the subparts, here are a few tools you can use:

- You run GDB once, and discover that the address of the RIP of `third_wheel` is `0xffffcd84`.
- For your inputs, you may use `SHELLCODE` as a 100-byte shellcode.
- The number `0` [illegible] at address `0x8048773`. The number `0xe4ff` is interpreted as `jmp *esp`
- If needed, you [illegible] as `OUTPUT`, slicing it using Python 2 syntax.

Assume that:

- You are on a [illegible] system.
- There is no other compiler padding or saved additional registers.
- `main` calls `valentine` with the appropriate arguments.
- **Stack canaries** are enabled and no other memory safety defenses are enabled.
- The stack canary is four completely random bytes (**no null byte**).
- `read_input(buf)` is a memory-safe function that writes to `buf` without any overflows.

Write your exploits in Python 2 syntax (just like in Project 1).

Q2.1 Fill in the following stack diagram, assuming that the program is paused at **Line 14**. Each row should contain a struct member, local variable, the SFP of `third_wheel`, or canary (the value in each row does not have to be four bytes long).

**Stack**

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |

Q2.2 In the first call to `third_wheel`, we want to leak the value of the stack canary. What should be the missing values at line 21 in order to make this exploit possible?

*Provide a decimal integer in each box.*

For the rest of the question, assume that **ASLR** is enabled in addition to stack canaries. Assume that the code section of memory has not been randomized.

Q2.3 Provide an input to each of the lines below in order to leak the stack canary in the first call to `third_wheel`. If you don't need an input, you must write "Not Needed".

Provide a string value for `tape[0]`:

Provide an input to `fgets` in `third_wheel`:

Q2.4 Provide an input to each of the lines below in order to run the malicious shellcode in the second call to `third_wheel`. If you don't need an input, you must write "NotNeeded".

Provide a string value for `tape[1]`:

Provide an inp wheel: