CS 161 Spring 2025 — Introduction to Computer Security — Discussion 2

Question 1 *Software Vulnerabilities*

For the following code, assume an attacker can control the value of `basket`, `n`, and `owner_name` passed into `search_basket`.

This code contains several security vulnerabilities. **Circle *three* such vulnerabilities** in the code and briefly explain each of the three on the next page.

```c
struct cat {
    char name[64];
    char owner[64];
    int age;
};

/* Searches through a BASKET of cats of length N (N should be less than 32). Adopts all cats with age less than 12 (kittens). Adopted kittens have their owner name overwritten with OWNER_NAME. Returns the number of kittens adopted. */
size_t search_basket(struct cat *basket, int n, char *owner_name) {
    struct cat kittens[32];
    size_t num_kittens = 0;
    if (n > 32) return -1;
    for (size_t i = 0; i <= n; i++) {
        if (basket[i].age < 12) {
            /* Reassign the owner name. */
            strcpy(basket[i].owner, owner_name);
            /* Copy the kitten from the basket. */
            kittens[num_kittens] = basket[i];
            num_kittens++;
            /* Print helpful message. */
            printf("Adopting kitten: ");
            printf(basket[i].name);
            printf("\n");
        }
    }
    /* Adopt kittens. */
    adopt_kittens(kittens, num_kittens); // Implementation not shown.
    return num_kittens;
}
```

1. Explanation:

2. Explanation:

3. Explanation:

Describe how an attacker could exploit these vulnerabilities to obtain a shell:

Question 2 *Hacked EvanBot*

Hacked EvanBot is running code to violate students' privacy, and it's up to you to disable it before it's too late!

```
 1 #include <stdio.h>
 2 
 3 void spy_on_students(void) {
 4     char buffer[16];
 5     fread(buffer, 1, 24, stdin);
 6 }
 7 
 8 int main() {
 9     spy_on_students();
10     return 0;
11 }
```

The shutdown code for Hacked EvanBot is located at address `0xdeadbeef`, but there's just one problem—Bot has learned a new memory safety defense. Before returning from a function, it will check that its saved return address (rip) is not `0xdeadbeef`, and throw an error if the rip is `0xdeadbeef`.

Clarification during exam: Assume little-endian x86 for all questions.

Assume all x86 instructions are 8 bytes long. Assume all compiler optimizations and buffer overflow defenses are disabled.

The address of `buffer` is `0xbffff110`.

Q2.1 (3 points) In the next 3 subparts, you'll supply a malicious input to the `fread` call at line 5 that causes the program to execute instructions at `0xdeadbeef`, *without* overwriting the rip with the value `0xdeadbeef`.

The first part of your input should be a single assembly instruction. What is the instruction? x86 pseudocode or a brief description of what the instruction should do (5 words max) is fine.

Q2.2 (3 points) The second part of your input should be some garbage bytes. How many garbage bytes do you need to write?

○ (G) 0 ○ (H) 4 ○ (I) 8 ○ (J) 12 ○ (K) 16 ○ (L) ——

Q2.3 (3 points) What are the last 4 bytes of your input? Write your answer in Project 1 Python syntax, e.g. `\x12\x34\x56\x78`.

Q2.4 (3 points) When does your exploit start executing instructions at `0xdeadbeef`?

○ (G) Immediately when the program starts

○ (H) When the `main` function returns

○ (I) When the `spy_on_students` function returns

○ (J) When the `fread` function returns

○ (K) ——

○ (L) ——

Question 3 ***I Understood that Reference!***

Consider the following vulnerable C code:

```c
void vulnerable(int start, char *ptr) {
    ptr[start] = ptr[3];
    ptr[start + 1] = ptr[2];
    ptr[start + 2] = ptr[1];
    ptr[start + 3] = ptr[0];
}

void helper(int8_t num) {
    if (num > 124) {
        return;
    }
    char arr[128];
    fgets(arr, 128, stdin);
    vulnerable(num, arr);
}

int main(void) {
    int y;
    fread(&y, sizeof(int), 1, stdin);
    helper(y);
    return 0;
}
```

Assume that:

- You are on a little-endian 32-bit x86 system.
- There is no other compiler padding or saved additional registers.

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

Q3.1 (3 min) Fill in the stack diagram below, assuming that execution has entered the call to `vulnerable`:

RIP of `main`
SFP of `main`
RIP of `vulnerable`
SFP of `vulnerable`

For the rest of this question, assume that the RIP of `main` is located at `0xbfffdc0c` and that your malicious shellcode is located at `0xef302010`.

In the next two subparts, construct an exploit that executes your malicious shellcode.

Q3.2 (5 min) Provide an input to the variable `y` in the `fread` in `main`.

For this subpart only, you may write a decimal number instead of its byte representation.

Q3.3 (5 min) Provide an input to the variable `arr` in the `fgets` in `helper`.