Software Security

Question 1 ***Software Vulnerabilities***

For the following code, assume an attacker can control the value of `basket` passed into `search_basket`. The value of `n` is constrained to correctly reflect the number of cats in `basket`.

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

```
1  struct cat {
2      char name[1024];
3      int age;
4  };
5
6  /* Searches through a basket of cats of size at most 32.
7     Returns the number of kittens or -1 if there are no kittens */
8  int search_basket(struct cat basket[], size_t n) {
9      struct cat kittens[32];
10     char kitten_names[1024], cmd[1024];
11     int i, total_kittens = 0, names_size = 0;
12
13     if (n > 32) return -1;
14
15     for (i = 0; i <= n; i++) {
16         if (basket[i].age < 12) {
17             size_t len = strlen(basket[i].name);
18             snprintf(kitten_names + names_size, len, "%s ", basket[i].name);
19             kittens[total_kittens] = basket[i];
20             names_size += len;
21             total_kittens += 1;
22         }
23     }
24
25     if (total_kittens > 5) {
26             const char *fmt = "adopt-kittens --num_kittens %d --names %s";
27             snprintf(cmd, sizeof cmd, fmt, total_kittens, kitten_names);
28             system(cmd);
29     }
30     return total_kittens;
31 }
```

Reminders:

- **snprintf(buf, len, fmt, ...)** works like **printf**, but instead writes to **buf**, and won't write more than **len - 1** characters. It terminates the characters written with a '\0'.
- **system** runs the shell command given by its first argument.

1. Explanation:

2. Explanation:

3. Explanation:

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

Question 2 ***C Memory Defenses***

Mark the following statements as True or False and justify your solution. Please feel free to discuss with students around you.

1. Stack canaries completely prevent a buffer overflow from overwriting the return instruction pointer.

2. A format-string vulnerability can allow an attacker to overwrite values below the stack pointer

3. An attacker exploits a buffer overflow to redirect program execution to their input. This attack no longer works if the data execution prevention/executable space protection/NX bit is set.

4. If you have a non-executable stack and heap, buffer overflows are no longer exploitable.

5. If you use a memory-safe language, some buffer overflow attacks are still possible.

6. ASLR, stack canaries, and NX bits all combined are insufficient to prevent exploitation of all buffer overflow attacks.

Short answer!

1. What vulnerability would arise if the canary was above the return address?

2. What vulnerability would arise if the stack canary was between the return address and the saved frame pointer?

3. Assume ASLR is enabled. What vulnerability would arise if the instruction **jmp *esp** exists in memory?

Question 3 ***TCB (Trusted Computing Base)***

In lecture, we discussed the importance of a TCB and the thought that goes into designing it. Answer these following questions about the TCB:

1. What is a TCB?

   _____

2. What can we do to reduce the size of the TCB?

   _____

3. What components are included in the (physical analog of) TCB for the following security goals:

   (a) Preventing break-ins to your apartment

   _____

   (b) Locking up your bike

   _____

   (c) Preventing people from riding BART for free

   _____

   (d) Making sure no explosives are present on an airplane

   _____