Question 1 *Software Vulnerabilities* (20 min)

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

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

```c
1  struct food {
2    char name[1024];
3    int calories;
4  };
5
6  /* Evaluate a shopping basket with at most 32 food items.
7       Returns the number of low-calorie items, or -1 on a problem. */
8  int eval_basket(struct food basket[], size_t n)
9  {
10    struct food good[32];
11    char bad[1024], cmd[1024];
12    int i, total = 0, ngood = 0, size_bad = 0;
13
14    if (n > 32)
15      return -1;
16
17    for ( i = 0; i <= n; ++i ) {
18      if (basket[i].calories < 100)
19        good[ngood++] = basket[i];
20      else if (basket[i].calories > 500) {
21        size_t len = strlen(basket[i].name);
22        snprintf(bad + size_bad, len, "%s ", basket[i].name);
23        size_bad += len;
24      }
25
26      total += basket[i].calories;
27    }
28
29    if (total > 2500) {
30      const char *fmt = "health-factor --calories %d --bad-items %s";
31      fprintf(stderr, "lots of calories!");
32      snprintf(cmd, sizeof cmd, fmt, total, bad);
33      system(cmd);
34    }
35
36    return ngood;
37  }
```

Reminder: **strlen** calculates the length of a string, not including the terminating $'\backslash 0'$ character. **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 $'\backslash 0'$. **system** runs the shell command given by its first argument.