Song
Spring 2015

CS 161 Computer Security

Discussion 4

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.

```
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.

Solution: Solution: There are significant vulnerabilities at lines **17**/**19**,**22**, and **33**.

Line **17** has a fencepost error: the conditional test should be $i < n$ rather than $i <= n$. The test at line **14** assures that **n** doesn't exceed 32, but if it's equal to 32, and if all of the items in **basket** are "good", then the assignment at line **19** will write past the end of **good**, representing a buffer overflow vulnerability.

At line **22**, there's an error in that the length passed to **snprintf** is *supposed* to be available space in the buffer (which would be **sizeof bad - size_bad**), but instead it's the length of the string being copied (along with a blank) into the buffer. Therefore by supplying large names for items in **basket**, the attacker can write past the end of **bad** at this point, again representing a buffer overflow vulnerability.

At line **33**, a shell command is run based on the contents of **cmd**, which in turn includes values from **bad**, which in turn is derived from input provided by the attacker. That input could include shell command characters such as pipes ('|') or command separators (';'), facilitating *command injection*.

Some more minor issues concern the **name** strings in **basket** possibly not being correctly terminated with $'\backslash 0's$, which could lead to reading of memory outside of **basket** at line **21** or line **22**.

Note that there are no issues with format string vulnerabilities at any of lines **22**, **31**, or **32**. For each of those, the format itself does not include any elements under the control of the attacker.

A final note: do not hesitate to ask for help! Our office hours exist to help you. Please visit us if you have any questions or doubts about the material.