CSCD437 Homework 3

Ethan Tuning

Sunny Sunheim likes writing C code. That's because when he writes it, the sun is always shining. If his code compiles, he won the battle and it's sunny. If his program runs properly with his input, he won the war and it's sunny *and* warm. Sunny smiles broadly when he wins the battle and the war, and then goes out into the sun for some serious R and R. Your job is to rain on Sunny's 'shining' C code and identify how it can be exploited (or that it can't be exploited – maybe Sunny got lucky (the sun was shining on him)). Show Sunny that he spends too much time in the sun!

For each of the following letters, identify problems/vulnerabilities and suggest how to avoid them (first) inside the function itself or (second - assuming first is not possible) how you could modify the parameters to the function to help. Since you have the code, compile and run to confirm any suspicions. All problems assume a command line argument is specified when the program is run.

Problem a:

```
1   int foo(char *arg, char *out){
2
3     strcpy(out, arg);
4     return 0;
5   }
6
7   int main(int argc, char *argv[]){
8
9     char buf[64];
10
11    if (argc != 2){
12      fprintf(stderr, "a: argc != 2\n");
13      exit(EXIT_FAILURE);
14    }
15    foo(argv[1], buf);
16    return 0;
17  }
```

Answer a:

In the foo() function strcpy() is used blindly. As it just copies bytes from arg into the out without verifying anything. This is bad because the input could be bigger than the buffer, causing an overflow. To fix this we could use strncpy() so we could specify the length that we need.

Problem b:

```
1   int foo(char *arg){
2
3     char buf[128];
4     int len, i;
5     len = strlen(arg);
6
7     if(len > 136)
8       len = 136;
9
10    for(i = 0; i <= len; i++)
11      buf[i] = arg[i];
12
13    return 0;
14  }
15
16  int main(int argc, char *argv[]){
17
18    if (argc != 2){
19      fprintf(stderr, "b: argc != 2\n");
20      exit(EXIT_FAILURE);
21    }
22    foo(argv[1]);
23    return 0;
24  }
```

Answer b:

In this one there are a few problems. For one the buffer is a different size than then the input string. This can cause, yet again, overflow. For this issue we can fix the numbers so that they match. Next, the buffer is not initialized. To fix that we just initialize the buffer. Lastly, the for loop needs to stop at $<$ and not as it is now $<=$. So that would be a problem because then an extra byte would be copied into the buffer.

Problem c:

```
1   int bar(char *arg, char *targ, int ltarg){
2
3     int len, i;
4     len = strlen(arg);
5
6     if(len > ltarg)
7       len = ltarg;
8
9     for(i = 0; i <= len; i++)
10      targ[i] = arg[i];
```

```
11
12        return 0;
13      }
14
15      int foo(char *arg){
16
17        char buf[128];
18        bar(arg, buf, 140);
19        return 0;
20      }
21
22      int main(int argc, char *argv[]){
23
24        if (argc != 2){
25          fprintf(stderr, "c: argc != 2\n");
26          exit(EXIT_FAILURE);
27        }
28        foo(argv[1]);
29        return 0;
30      }
```

Answer c:

As with the last problem the buffer and the input are different sizes. Again to avoid overflow we need to fix this. Also we need to initialize the buffer. Last, strlen() is used to determine input length. This is a problem because the input could be bigger than what is returned by strlen(). To fix this, use a loop so that it will recall the exact amount of elements that you have for your buffer. This will eliminate the overflow issue.

Problem d:

```
1       int foo(char *arg, short arglen){
2
3         char buf[1024];
4         int i, maxlen = 1024;
5
6         if (arglen < maxlen){
7
8           for (i = 0; i < strlen(arg); i++)
9             buf[i] = arg[i];
10
11        }
12        return 0;
13      }
```

```
14
15      int main(int argc, char *argv[]){
16
17        if (argc != 2){
18          fprintf(stderr, "d: argc != 2\n");
19          exit(EXIT_FAILURE);
20        }
21
22        foo(argv[1], strlen(argv[1]));
23        return 0;
24      }
```

Answer d:

Again need to initialize buffer. Also we need to NOT use strlen() the two times that it is used in this program. For all of the reasons stated earlier. There is to much potential to have overflow from strlen(). Also in foo() eliminate the length parameter.