Quiz 3

Sunday, November 10, 2019 11:37 PM

Ryan Holland
Quiz 3
Fuzzing practice
11/10/19

First, I read the program. I filled out the input functions with code which generates random ASCII characters in the range [0…126]. Here they are:

```
// Generates a random ASCII character that can possibly trigger the fail
conditions in testme().
char inputChar()
{
    // random [0 ... 126]
    int charNum;
    charNum = (int)rand() * 94 + 32;

    return charNum;
}

// Generates a random ASCII character array that can possibly trigger
the fail conditions in testme().
// We need at least 6 chars in the array since some of the conditions
check its index 5.
char arr[6];
char *inputString()
{
    int charNum, i;
    for (i = 0; i < 6; i++) {
        // random [0 ... 126]
        charNum = (int)rand() * 94 + 32;
        arr[i] = charNum;
    }

    return arr;
}
```

I tested it on flip. It returns a lot of unprintable characters, but also some of the desired ones. However, I ran it about 13 million times, and it didn't exit.

if C is 1/(char range), it is a 1/126 chance of being the desired character. state never decreases. So it should take 126 rolls of the dice to reach the desired character and raise state by 1. Then that has to happen 8 more times. 126*8 = 1,008 rolls on average. That should take a few seconds. But afterwards, for the array to say "reset\0", it has to line up 6 characters out of the 126. $126^6 = 4x10^{12}$ rolls. So I need

to narrow down my input array.

Also, I had bigger problems. I noticed state had never increased. So I tested it by setting c to 91 (should be character "["), and also testing whether char=0 outputs a "\0" null terminator. The experiment was a success. So I must be doing the rand num gen wrong.

Since the assignment says I can choose the input pool, and we are going for speed, I could simply include only the chars that trigger program conditions. That seems like it's against the spirit of the assignment, but then again, aren't we supposed to narrow our random inputs down to realistic values? I could also just include the lower case numbers and the brackets, and the null. The null adds a problem since it's #0, not next to the others. I can code it, it just seems a little unclean.

after googling it, I realized the rand() function in c is not generating a random number in [0…1], so I found a wrapper function on stack overflow.

```
// Not having a function like this built in is why I don't like using c.
It's like coding in the stone age. But I guess c is worth it if you have
the need for speed.
// source: https://stackoverflow.com/questions/6218399/how-to-generate-
a-random-number-between-0-and-1
int randInRange(int min, int max)
{
    return min + (int)(rand() / (double)(RAND_MAX + 1) * (max - min +
    1));
}

// Generates a random ASCII character that can possibly trigger the fail
conditions in testme().
char inputChar()
{
    int num;
    num = randInRange(91, 127);
    if (num == 127) {
        num = 0;
    }

    return num;
}


// Generates a random ASCII character array that can possibly trigger
the fail conditions in testme().
// We need at least 6 chars in the array since some of the conditions
check its index 5.
char arr[6];
char *inputString()
{
    int num, i;
    for (i = 0; i < 6; i++) {
        num = randInRange(91, 127);
        if (num == 127) {
            num = 0;
        }
        arr[i] = num;
```

```
    }

    return arr;
}
```

However, that didn't work, and I realized I had various problems. Several of the characters were spread out in the ASCII table. The random int generator was not even compiling.
I decided to iterate through only the desired chars. Limiting our inputs sounds smart. I tried a few ways, and it seemed cleanest to give it a list of ints.

```
//char *charSet = "[({ ax})]rest";
int charList[15] = { 0, 40, 41, 32, 91, 93, 123, 124, 125, 97, 120, 114,
101, 115, 116 };

int randInRange(int min, int max)
{
    int num;
    num = rand() % (max - min + 1) + min;
    return num;
}


// Generates a random ASCII character that can possibly trigger the fail
conditions in testme().
char inputChar()
{
    int num;
    num = randInRange(0, 13); // 13 would be null terminator
    return charList[num];
}


// Generates a random ASCII character array that can possibly trigger
the fail conditions in testme().
// We need at least 6 chars in the array since some of the conditions
check its index 5.
char arr[6];
char *inputString()
{
    int num, i;
    for (i = 0; i < 6; i++) {
        arr[i] = inputChar();
    }

    return arr;
}
```

It was getting to state 9 really quickly, but not generating RESET.

With 15 inputs, and 5 slots, we have a one in 15^5 = 1/11,390,625 chance of getting "reset\0". Unless I'm

really bad at math. That's only a few minutes' time. But I'm not sure that null is generating.

Then I realized something cool. s[5] will be null all the time if that's the end of the string. And that will shorten things.
14 ^ 4 = 38,416
so that's great. But after iterating 2 million times, it didn't trigger. I tested to make sure it can trigger on null. And I lowered the array size which I'd forgotten to do earlier. At long last, it triggered:

Iteration 1044941: c = (, s = xsa{{, state = 9
Iteration 1044942: c = s, s = t{[)a, state = 9
Iteration 1044943: c = r, s =)e[xe, state = 9
Iteration 1044944: c =], s = { e}s, state = 9
Iteration 1044945: c = s, s =)[{{ , state = 9
Iteration 1044946: c = |, s = {e]s(, state = 9
Iteration 1044947: c = }, s = **reset**, state = 9
error flip1 ~/362/random 198%

A second test was much quicker:

Iteration 2805: c = x, s = {x(}, state = 9
Iteration 2806: c =], s = s]x[[, state = 9
Iteration 2807: c = {, s = |a|||, state = 9
Iteration 2808: c = {, s = t] [|, state = 9
Iteration 2809: c = t, s = s})]e, state = 9
Iteration 2810: c = e, s = t[]e{, state = 9
Iteration 2811: c = x, s =)te[, state = 9
Iteration 2812: c =), s =)}}}t, state = 9
Iteration 2813: c = t, s = rs[rt, state = 9
Iteration 2814: c = }, s = reset, state = 9
error flip1 ~/362/random 201%

So I had to remember how to use gcov. The manpage was actually the most help.
https://linux.die.net/man/1/gcov

flip1 ~/362/random 205% gcov random
File 'random.c'
Lines executed:**97.30% of 37**
Creating 'random.c.gcov'