A SYMBOLIC MEMORY ADDRESSING EXAMPLE

Listing 1 shows a concrete example of memory addressing problem in symbolic execution. Let's assume an initial seed with the hexadecimal data `02000000FFFF02000000FFFF`, where makes `len1` and `len2` both equal to 2 and the program prints out "Small data". For symbolic execution, constraint shown in Equation 1 will be generated before solving the condition branch at line 20 for exploring new paths.

Listing 1: Example of mistake handling of symbolic memory addressing.

```
1  char *data = (char *) malloc(100);
2  read(0, data, 100);
3  char *pos = data;
4
5  int len1 = *(int *)pos;
6  pos += sizeof(int);
7  // solving position 1
8  if (!(len1 > 0 && len1 <10)) exit(-1);
9  // data reading with the length of `len1`.
10 pos += len1;
11
12 int len2 = *(int *)pos;
13 pos += sizeof(int);
14 // solving position 2
15 if (!(len2 > 0 && len2 <10)) exit(-1);
16 // data reading with the length of `len2`.
17 pos += len2;
18
19 // solving position 4
20 if (len1 + len2 > 5) printf("Big data\n");
21 else printf("Small data\n");
```

$$\begin{gathered} input[0:4] > 0 \\ input[0:4] < 10 \\ input[6:10] > 0 \\ input[6:10] < 10 \\ input[0:4] + input[6:10] > 5 \end{gathered} \tag{1}$$

After receiving the solving result from SMT solver, a new seed with its data from offset 0 to 3 and 6 to 9 modified is generated, and let us assume the generated seed as `03000000FFFF03000000FFFF`. The seed can theoretically lead to the output of "Big data", but when parsing the data block, `len2` will be assigned a value of `0xFF000000`, which will cause the program to exit at line 15.