Lab 4 Report
CS260-001: Computer Security
Prof. Heng Yin

Abhishek Srivastava
Student Id: 861307778
February 14, 2017

Experimenting with Evolutionary Fuzzing

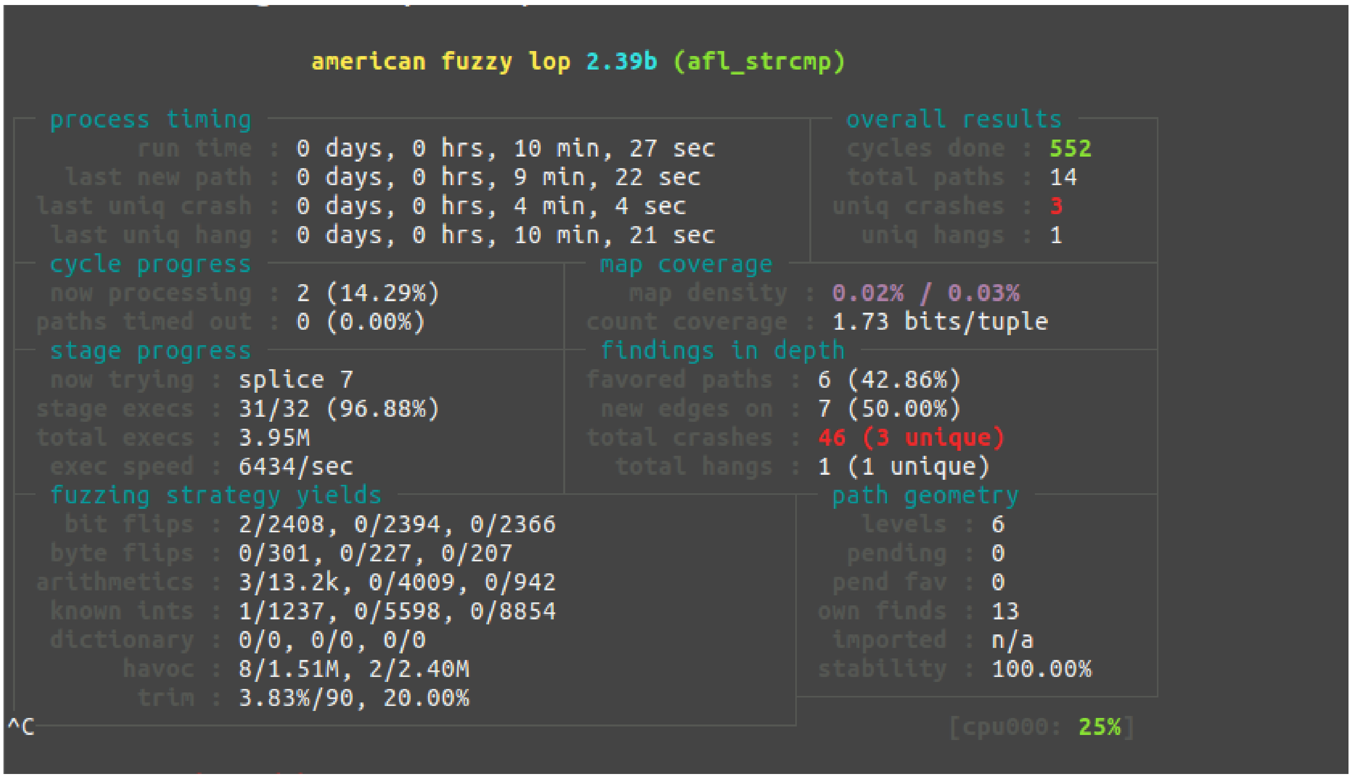
Problem 1

Does AFL crash the program? Explain why do you think AFL can or cannot crash it?

Solution

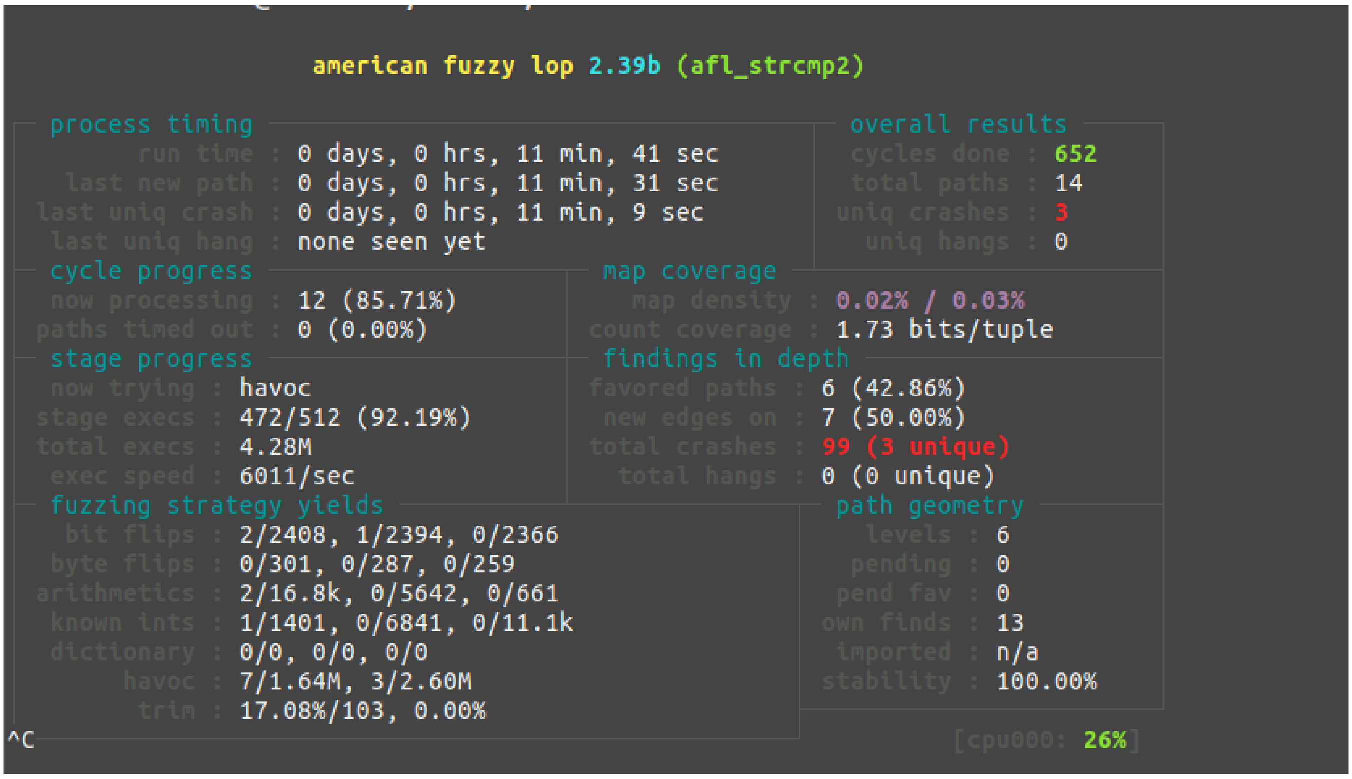
Yes, the AFL program does crashes both of the program ran with it and it generated 3 unique crashes and covered 14 total control paths as well for each case. AFL is fuzzing tool and it generates different test cases based on the seeded file which is being provided initially. The main focus of AFL is to generate such cases so that it can cover as many possible paths as possible in the program. It applies multiple kind of operation on the input test case such as: **flip1**, **havoc**, **arith8** and **slice** to generate fuzzy inputs and observe how program behaves.

Figure 1 and Figure 2 shows the result of running AFL for about 10 mins.



```
                    american fuzzy lop 2.39b (afl_strcmp)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 10 min, 27 sec      │  cycles done : 552    │
│   last new path : 0 days, 0 hrs, 9 min, 22 sec       │  total paths : 14     │
│ last uniq crash : 0 days, 0 hrs, 4 min, 4 sec        │ uniq crashes : 3      │
│  last uniq hang : 0 days, 0 hrs, 10 min, 21 sec      │   uniq hangs : 1      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 2 (14.29%)        │    map density : 0.02% / 0.03%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.73 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : splice 7              │ favored paths : 6 (42.86%)             │
│ stage execs : 31/32 (96.88%)        │  new edges on : 7 (50.00%)             │
│ total execs : 3.95M                 │ total crashes : 46 (3 unique)          │
│  exec speed : 6434/sec              │  total hangs : 1 (1 unique)            │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 2/2408, 0/2394, 0/2366                │    levels : 6          │
│  byte flips : 0/301, 0/227, 0/207                   │   pending : 0          │
│ arithmetics : 3/13.2k, 0/4009, 0/942                │  pend fav : 0          │
│  known ints : 1/1237, 0/5598, 0/8854                │ own finds : 13         │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│       havoc : 8/1.51M, 2/2.40M                      │ stability : 100.00%    │
│        trim : 3.83%/90, 20.00%                      ├────────────────────────┘
^C────────────────────────────────────────────────────┘          [cpu000: 25%]
```

Figure 1: Screen Shot of running AFL for program afl_strcmp.



```
                    american fuzzy lop 2.39b (afl_strcmp2)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 11 min, 41 sec      │  cycles done : 652    │
│   last new path : 0 days, 0 hrs, 11 min, 31 sec      │  total paths : 14     │
│ last uniq crash : 0 days, 0 hrs, 11 min, 9 sec       │ uniq crashes : 3      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 12 (85.71%)       │    map density : 0.02% / 0.03%        │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.73 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                 │ favored paths : 6 (42.86%)            │
│ stage execs : 472/512 (92.19%)      │  new edges on : 7 (50.00%)            │
│ total execs : 4.28M                 │ total crashes : 99 (3 unique)         │
│  exec speed : 6011/sec              │  total hangs : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 2/2408, 1/2394, 0/2366                │    levels : 6          │
│  byte flips : 0/301, 0/287, 0/259                   │   pending : 0          │
│ arithmetics : 2/16.8k, 0/5642, 0/661                │  pend fav : 0          │
│  known ints : 1/1401, 0/6841, 0/11.1k               │ own finds : 13         │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│       havoc : 7/1.64M, 3/2.60M                      │ stability : 100.00%    │
│        trim : 17.08%/103, 0.00%                     ├────────────────────────┘
^C────────────────────────────────────────────────────┘          [cpu000: 26%]
```

Figure 2: Screen Shot of running AFL for program afl_strcmp2.

Problem 2

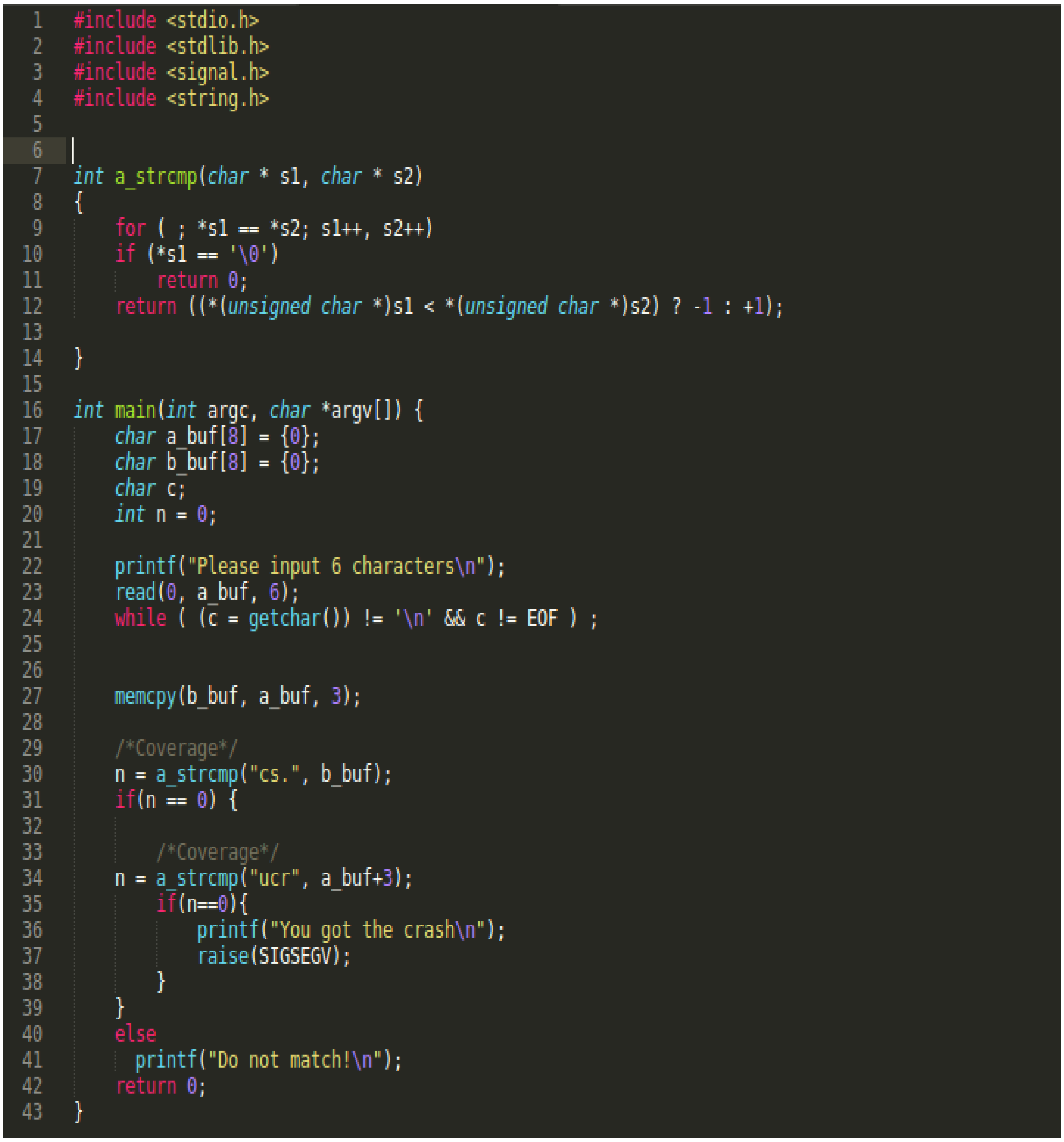
Look at the seed files (ending with +cov) in the queue. These inputs increase the branch coverage according to AFL. For each of these seeds, explain what branch coverage it causes. You can use the Line numbers in the source file.

Solution

afl_strcmp: Figure 3 shows the source code of the programs taken as reference for line numbers. Original Text was : "abABcdCD". Table 1 shows the seed file names, their content and their branch coverage. I am not considering while loop to take input as a branch. Also some coverage path are coming as same because I am not including the loop repetition over the same branch. Branch at 12 takes two routes so different routes will generate different coverage. I have marked the arrows to show what branches are covered at what line number.

Table 1: Output of seed files for test afl_strcmp file.

Seed Name	Seed Input	Branch Coverage
id:000001,src:000000,op:flip1,pos:0,+cov	cbABcdCD	$30 \rightarrow 9 \rightarrow 12 \rightarrow 40$
id:000007,src:000000,op:havoc,rep:16,+cov	dŜˆD̂@̂F̂@̂ÂÂd	$30 \rightarrow 12 \rightarrow 40$
id:000009,src:000001,op:arith8,pos:1,val:+17,+cov	csABcdCD	$30 \rightarrow 9 \rightarrow 12 \rightarrow 40$
id:000010,src:000009,op:arith8,pos:2,val:-19,+cov	cs.BcdCD	$30 \rightarrow 9 \rightarrow 10 \rightarrow 34 \rightarrow 12$
id:000012,src:000010,op:havoc,rep:2,+cov	cs.ucdCD	$30 \rightarrow 9 \rightarrow 10 \rightarrow 34 \rightarrow 9 \rightarrow 12$
id:000013,src:000012,op:flip1,pos:4,+cov	cs.u?dCD	$30 \rightarrow 9 \rightarrow 10 \rightarrow 34 \rightarrow 9 \rightarrow 12$



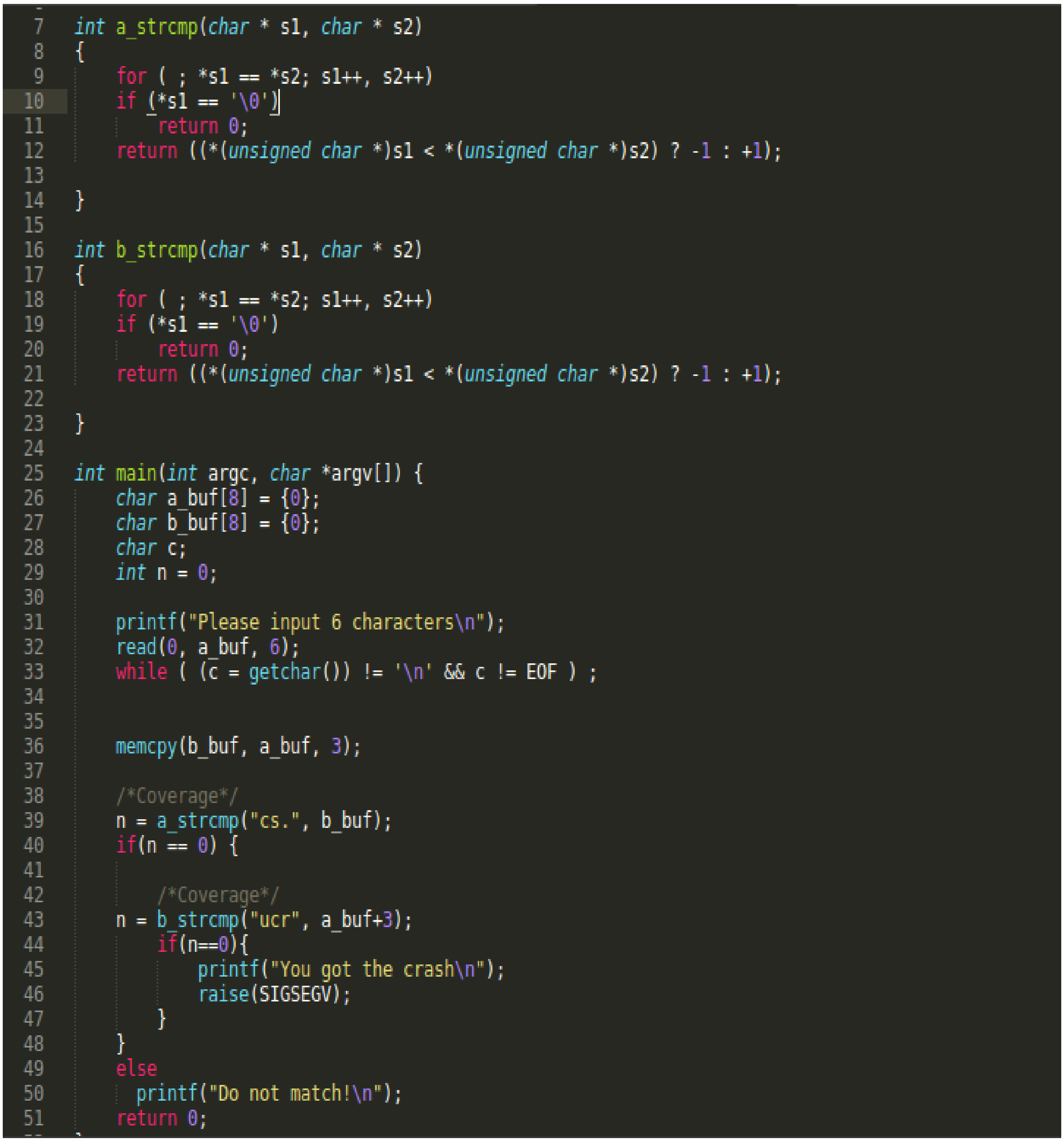
```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>


int a_strcmp(char * s1, char * s2)
{
    for ( ; *s1 == *s2; s1++, s2++)
    if (*s1 == '\0')
        return 0;
    return ((*(unsigned char *)s1 < *(unsigned char *)s2) ? -1 : +1);

}

int main(int argc, char *argv[]) {
    char a_buf[8] = {0};
    char b_buf[8] = {0};
    char c;
    int n = 0;

    printf("Please input 6 characters\n");
    read(0, a_buf, 6);
    while ( (c = getchar()) != '\n' && c != EOF ) ;


    memcpy(b_buf, a_buf, 3);

    /*Coverage*/
    n = a_strcmp("cs.", b_buf);
    if(n == 0) {

        /*Coverage*/
    n = a_strcmp("ucr", a_buf+3);
        if(n==0){
            printf("You got the crash\n");
            raise(SIGSEGV);
        }
    }
    else
      printf("Do not match!\n");
    return 0;
}
```

Figure 3: Code for the program afl_strcmp.

afl_strcmp2: Figure 4 shows the source code of the program taken as reference for line numbers. Original Text input was : “abABcdCD”. Table 2 shows the seed file names, their content and there branch coverage. I have taken same assumptions as the previous program. Branch at 12 and 21 takes two routes so different routes will generate different coverage.

Table 2: Output of seed files for test afl_strcmp2 file.

Seed Name	Seed Input	Branch Coverage
id:000001,src:000000,op:flip1,pos:0,+cov	cbABcdCD	$39 \rightarrow 9 \rightarrow 12 \rightarrow 49$
id:000008,src:000000,op:havoc,rep:64,+cov	G$^{@@}$[[$^{@}d$	$39 \rightarrow 12 \rightarrow 49$
id:000009,src:000001,op:arith8,pos:1,val:+17,+cov	csABcdCD	$39 \rightarrow 9 \rightarrow 12 \rightarrow 49$
id:000010,src:000009,op:arith8,pos:2,val:-19,+cov	cs.BcdCD	$39 \rightarrow 9 \rightarrow 10 \rightarrow 43 \rightarrow 21$
id:000012,src:000010+000002,op:splice,rep:2,+cov	cs.u??	$39 \rightarrow 9 \rightarrow 10 \rightarrow 43 \rightarrow 18 \rightarrow 21$
id:000013,src:000012,op:havoc,rep:2,+cov	cs.ucs?u	$39 \rightarrow 9 \rightarrow 10 \rightarrow 43 \rightarrow 18 \rightarrow 21$



```c
int a_strcmp(char * s1, char * s2)
{
    for ( ; *s1 == *s2; s1++, s2++)
    if (*s1 == '\0')
        return 0;
    return ((*(unsigned char *)s1 < *(unsigned char *)s2) ? -1 : +1);

}

int b_strcmp(char * s1, char * s2)
{
    for ( ; *s1 == *s2; s1++, s2++)
    if (*s1 == '\0')
        return 0;
    return ((*(unsigned char *)s1 < *(unsigned char *)s2) ? -1 : +1);

}

int main(int argc, char *argv[]) {
    char a_buf[8] = {0};
    char b_buf[8] = {0};
    char c;
    int n = 0;

    printf("Please input 6 characters\n");
    read(0, a_buf, 6);
    while ( (c = getchar()) != '\n' && c != EOF ) ;


    memcpy(b_buf, a_buf, 3);

    /*Coverage*/
    n = a_strcmp("cs.", b_buf);
    if(n == 0) {

        /*Coverage*/
    n = b_strcmp("ucr", a_buf+3);
        if(n==0){
            printf("You got the crash\n");
            raise(SIGSEGV);
        }
    }
    else
      printf("Do not match!\n");
    return 0;
```

Figure 4: Code for the program afl_strcmp2.

Problem 3

Pay attention the file names, they show how these inputs are mutated from their parents. Please explain the evolution of these seed files.

Solution

Each seed file follow the some pattern to generate the names for fuzzed inputs file names.

The original input name for (**id:000000,orig:test.txt**). **id** means seed id for this file is 000000, **orig** indicates that it is original file input and its name is *test.txt.*

For the seeded file name are like (**id:000013,src:000012,op:havoc,rep:2,+cov**). **id** means the seed id for this file is 000013, **src** means the source file id for generating this file is 000012, **op** means which operation it performed on the source file to generate the seed file. **rep** this field is operation dependent here it means number of repetitions done. **+cov** means it has performed additional coverage.