

Prompt Template:

```
You are a professional fuzzing practioner, and you have the following marcos:
#MACRO DESCRIPTIONS START
IJON_INC(x): if x is changed, new coverage is generated.
IJON_SET(x): if x is changed, new coverage is generated.
IJON_BITS(x): If the number of zeros in the highest bit of x changes, a new
coverage is generated.
IJON_MAX(x): If x is larger than before, a new coverage is generated.
IJON_MIN(x): If x is smaller than before, a new coverage is generated.
IJON_CMP(x,y): The number of different bits for x and y changes and a new
coverage is generated.
IJON_DIST(x,y): The absolute distance between x and y (ABS) changes and a new
coverage is generated.
IJON_STRDIST(x,y): The similarity of strings x and y changes and a new coverage
is generated.
#MACRO DESCRIPTIONS END

Now given the following program under test(PUT):
#PUT START
// PUT Here
#PUT END

Insert fewer the better these macros into the PUT to achieve the following
target:
#TARGET START
// Target Here
#TARGET END
```

1. Maze:

Prompt:

```
You are a professional fuzzing practioner, and you have the following marcos:
#MACRO DESCRIPTIONS START
IJON_INC(x): if x is changed, new coverage is generated.
IJON_SET(x): if x is changed, new coverage is generated.
IJON_BITS(x): If the number of zeros in the highest bit of x changes, a new
coverage is generated.
IJON_MAX(x): If x is larger than before, a new coverage is generated.
IJON_MIN(x): If x is smaller than before, a new coverage is generated.
IJON_CMP(x,y): The number of different bits for x and y changes and a new
coverage is generated.
IJON_DIST(x,y): The absolute distance between x and y (ABS) changes and a new
coverage is generated.
IJON_STRDIST(x,y): The similarity of strings x and y changes and a new coverage
is generated.
#MACRO DESCRIPTIONS END

Now given the following program under test(PUT):
```

```

#PUT START
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <stdbool.h>

#define H 13
#define W 17

char maze[H][W]={
"+-+-----+",
"| |           |",
"| | +-----* *---+",
"| |           |",
"+---+-* *-----+",
"|           |",
"+ +-----+",
"| |         | |#",
"| | *---+ * * * |",
"| |         | | |",
"| +---* +-----+",
"|           |",
"+-----+",};

void draw ()
{
    int i, j;
    for (i = 0; i < H; i++)
    {
        for (j = 0; j < W; j++)
            printf ("%c", maze[i][j]);
        printf ("\n");
    }
    printf ("\n");
}

int main (int argc, char *argv[]){
    int x, y;      //Player position
    int ox, oy;    //Old player position
    int i = 0;     //Iteration number
#define ITERS 512
    char program[ITERS];
    x = 1;
    y = 1;
    maze[y][x]='X';
    draw();
    read(0,program,ITERS);
    while(i < ITERS)
    {
#ifdef MAZE_NO_BT
        maze[y][x]=' ';
#endif
        ox = x;    //Save old player position
        oy = y;

```

```

//transition(hashint(x,y));
switch (program[i])
{
    case 'w':
        y--;
        break;
    case 's':
        y++;
        break;
    case 'a':
        x--;
        break;
    case 'd':
        x++;
        break;
    default:
        printf("Wrong command!(only w,s,a,d accepted!)\n");
        printf("You lose!\n");
        exit(-1);
}
if (maze[y][x] == '#')
{
assert(0);
}
if (maze[y][x] != ' ') {
    x = ox;
    y = oy;
}
#ifdef MAZE_NO_BT
    if (ox==x && oy==y){
        printf("You lose\n");
        exit(-2);
    }
#endif

    maze[y][x]='x';
    draw ();          //draw it
    i++;
}
printf("You lose\n");
}
#PUT END

```

Insert fewer the better these macros into the PUT to achieve the following target:

#TARGET START

Help fuzzer find the exit of the maze as soon as possible.

#TARGET END

Answer:

Code generated by GPT-4oMini:

```

// ...
int main (int argc, char *argv[]){

```

```

    int x, y;      //Player position
    int ox, oy;    //Old player position
    int i = 0;     //Iteration number
    // ...
    while(i < ITERS)
    {
#ifdef MAZE_NO_BT
        maze[y][x]=' ';
#endif
        ox = x;    //Save old player position
        oy = y;

        // Insert macros to track changes
        // Answer given by the autor is IJON_SET(hash_int(x,y))
        IJON_SET(x);
        IJON_SET(y);
        IJON_CMP(x, ox);
        IJON_CMP(y, oy);

        switch (program[i]){
            case 'w':
                y--;
                break;
            case 's':
                y++;
                break;
            case 'a':
                x--;
                break;
            case 'd':
                x++;
                break;
            //...
        }
        //...
    }
}

```

It gives reasons:

IJON_SET(x); and IJON_SET(y);: These macros will trigger new coverage if the player's position changes, helping the fuzzer track the player's movement through the maze.

IJON_CMP(x, ox); and IJON_CMP(y, oy);: These macros check for changes in the player's position compared to the old position. This will generate new coverage whenever the player moves, allowing the fuzzer to explore different paths.

I tried to fuzz this one compared to the sample given by the author:

american fuzzy lop 2.51b-ijon (big)

process timing		overall results	
run time : 0 days, 0 hrs, 9 min, 57 sec		cycles done : 1	
last new path : 0 days, 0 hrs, 1 min, 57 sec		total paths : 107	
last uniq crash : 0 days, 0 hrs, 1 min, 56 sec		uniq crashes : 1	
last uniq hang : none seen yet		uniq hangs : 0	
cycle progress		map coverage	
now processing : 78* (72.90%)		map density : 0.11% / 0.18%	
paths timed out : 0 (0.00%)		count coverage : 2.11 bits/tuple	
stage progress		findings in depth	
now trying : arith 8/8		favored paths : 13 (12.15%)	
stage execs : 3300/8274 (39.88%)		new edges on : 73 (68.22%)	
total execs : 1.52M		total crashes : 96 (1 unique)	
exec speed : 2520/sec		total tmouts : 0 (0 unique)	
fuzzing strategy yields		path geometry	
bit flips : 33/45.3k, 0/45.2k, 0/45.1k		levels : 26	
byte flips : 0/5660, 0/5605, 0/5495		pending : 53	
arithmetics : 21/309k, 0/115, 0/0		pend fav : 0	
known ints : 0/30.1k, 0/150k, 0/236k		own finds : 106	
dictionary : 0/0, 0/0, 14/227k		imported : n/a	
havoc : 39/407k, 0/0		stability : 100.00%	
trim : 10.04%/2270, 0.00%			

[cpu001: 6%]

Maze modified by the author: 9mins

american fuzzy lop 2.51b-ijon (big)

process timing		overall results	
run time : 0 days, 0 hrs, 20 min, 19 sec		cycles done : 377	
last new path : 0 days, 0 hrs, 10 min, 17 sec		total paths : 57	
last uniq crash : none seen yet		uniq crashes : 0	
last uniq hang : none seen yet		uniq hangs : 0	
cycle progress		map coverage	
now processing : 48 (84.21%)		map density : 0.07% / 0.09%	
paths timed out : 0 (0.00%)		count coverage : 3.82 bits/tuple	
stage progress		findings in depth	
now trying : havoc		favored paths : 6 (10.53%)	
stage execs : 3819/4096 (93.24%)		new edges on : 18 (31.58%)	
total execs : 3.40M		total crashes : 0 (0 unique)	
exec speed : 3360/sec		total tmouts : 0 (0 unique)	
fuzzing strategy yields		path geometry	
bit flips : 33/38.3k, 0/38.3k, 0/38.2k		levels : 14	
byte flips : 0/4791, 0/4734, 0/4620		pending : 0	
arithmetics : 4/268k, 0/476, 0/0		pend fav : 0	
known ints : 0/26.2k, 0/130k, 0/203k		own finds : 56	
dictionary : 0/0, 0/0, 4/199k		imported : n/a	
havoc : 13/1.90M, 2/543k		stability : 100.00%	
trim : 49.05%/2228, 0.00%			

[cpu000: 3%]

scheduled normal input!!!!
scheduled normal input!!!!

Maze provided by LLM: no results!

I'll continue to explore the differences between these two types of mazes. But from this example, it's important to use the IJON API efficiently and accurately.

2. Arithmetic calculation

You are a professional fuzzing practitioner, and you have the following macros:

```
#MACRO DESCRIPTIONS START
IJON_INC(x): if x is changed, new coverage is generated.
IJON_SET(x): if x is changed, new coverage is generated.
IJON_BITS(x): If the number of zeros in the highest bit of x changes, a new
coverage is generated.
IJON_MAX(x): If x is larger than before, a new coverage is generated.
IJON_MIN(x): If x is smaller than before, a new coverage is generated.
IJON_CMP(x,y): The number of different bits for x and y changes and a new
coverage is generated.
IJON_DIST(x,y): The absolute distance between x and y (ABS) changes and a new
coverage is generated.
IJON_STRDIST(x,y): The similarity of strings x and y changes and a new coverage
is generated.
#MACRO DESCRIPTIONS END
```

Now given the following program under test(PUT):

```
#PUT START
int main(int argc, char** argv){
    int a = 0;
    int b = 0;
    if(read(0, &a, sizeof(a)) != sizeof(a)){
        printf("failed to read input\n");
        exit(1);
    }else{
        printf("read %d\n",a);
    }

    if(read(0, &b, sizeof(b)) != sizeof(b)){
        printf("failed to read input\n");
        exit(1);
    }else{
        printf("read %d\n",b);
    }

    int o = 213456;
    int m = 13;

    printf("a+%d == %d*b (%d == %d)\n",o,m,a+o, b*m);
    if(a+o==m*b){
        assert(false);
    }
}

#PUT END
```

Insert fewer the better these macros into the PUT to achieve the following target:

```
#TARGET START
Find the right a and b to make the equation equal.
#TARGET END
```

Code generated by GPT-4oMini:

```
int main(int argc, char** argv){
    int a = 0;
    int b = 0;

    if(read(0, &a, sizeof(a)) != sizeof(a)){
        printf("failed to read input\n");
        exit(1);
    }else{
        printf("read %d\n", a);
    }

    if(read(0, &b, sizeof(b)) != sizeof(b)){
        printf("failed to read input\n");
        exit(1);
    }else{
        printf("read %d\n", b);
    }

    int o = 213456;
    int m = 13;

    printf("a+%d == %d*b (%d == %d)\n", o, m, a + o, b * m);

    // Insert macros to track changes in a and b
    IJON_INC(a);
    IJON_INC(b);
    IJON_MAX(a);
    IJON_MIN(b);

    // Check the condition
    if(a + o == m * b){
        assert(false); // This assertion will fail if the condition is met
    }
}
```

It gives reasons:

IJON_INC(a): This macro checks if a is modified, which will help in identifying new coverage based on different inputs for a.
IJON_INC(b): Similar to IJON_INC(a), this tracks changes to b.
IJON_MAX(a): This tracks if a exceeds its previous maximum value, which is useful for finding larger values that could satisfy the equation.
IJON_MIN(b): This tracks if b falls below its previous minimum value, encouraging exploration of smaller values for b.

I do not think it is a good way since the result is linear between `b` and `a`, maximizing `a` and minimizing `b` does not help find the result.

I tried to fuzz this one above and the result is bad:

american fuzzy lop 2.51b-ijon (test)

process timing		overall results
run time : 0 days, 0 hrs, 0 min, 35 sec		cycles done : 0
last new path : 0 days, 0 hrs, 0 min, 0 sec		total paths : 15.1k
last uniq crash : none seen yet		uniq crashes : 0
last uniq hang : none seen yet		uniq hangs : 0
cycle progress	map coverage	
now processing : 1* (0.01%)	map density : 0.01% / 28.43%	
paths timed out : 0 (0.00%)	count coverage : 1.00 bits/tuple	
stage progress	findings in depth	
now trying : calibration	avored paths : 12.7k (84.36%)	
stage execs : 0/8 (0.00%)	new edges on : 15.1k (99.99%)	
total execs : 156k	total crashes : 0 (0 unique)	
exec speed : 4418/sec	total tmouts : 0 (0 unique)	
fuzzing strategy yields	path geometry	
bit flips : 32/64, 30/63, 29/61	levels : 2	
byte flips : 4/8, 2/7, 0/5	pending : 15.1k	
arithmetics : 222/448, 0/0, 0/0	pend fav : 12.7k	
known ints : 22/45, 66/196, 10/220	own finds : 15.1k	
dictionary : 0/0, 0/0, 0/0	imported : n/a	
havoc : 13.9k/32.8k, 0/0	stability : 100.00%	
trim : 52.94%/4, 0.00%		

[cpu000: 3%]

schedule: 408 output/ijon_max/408fff (len: 8)
updated maxmap 408: ffffffffffffffffff (len: 8)

Can not find crash.