Assignment 4: Program Analysis

First Bug

Proof:

Given input `v6€ôÿá` , it causes a floating point exception when the program crashes

```
AddressSanitizer:DEADLYSIGNAL
=================================================================
==17459==ERROR: AddressSanitizer: FPE on unknown address 0x0000004c43a5 (pc 0x0000004c43a5 bp 0x7fffffffeded0 sp 0x7fffffffedea0 T0)
    #0 0x4c43a5 in sp_calloc /mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/./spalloc.c:237:8
    #1 0x4c4be0 in main /mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/driver.c:53:20
    #2 0x7fffff4270b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #3 0x41b31d in _start (/mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/fuzzed_dir/default/crashes/a.out+0x41b31d)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: FPE /mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/./spalloc.c:237:8 in sp_calloc
==17459==ABORTING
```

the bug is a case of `calloc()` being affected by the `large_alloc()` function, where it crashes when it exceeds a certain size limit. My guess is that the pagesize is not correctly allocated in the `large_alloc()` function, and by fixing this using the correct defined PAGESIZE, which assuming its 4096, running `afl-fuzz` again with -DSP_FAKE using the correct malloc library does not cause any crashes with floating point exceptions.

Driver:

```
55      /* simulating paths with hashed value */
56      switch(hash){
57          case 0:
58              printf("----------------------------- case 0 ---------------------------------\n");
59
60              var1 = sp_malloc(large_input);
61              var1 = sp_calloc(large_input, input);
62              printf("case 0 var 1 : %s \n", var1);
63              printf("case 0 var 2 : %s \n", var2);
64              // sp_free(singleLine[0]);
65              break;
```

Fix:

```c
// 4030 should be PAGESIZE
void *large_alloc(uint64_t size) {
  if (size > ~0ULL-3*PAGESIZE)
    return NULL;
  uint64_t pages = (size + PAGESIZE)>>PAGESHIFT; // bug 2 fix 4096 -> PAGESIZE
  void * rv = mmap(NULL,
                   (pages + 2) << PAGESHIFT,
                   PROT_READ|PROT_WRITE,
                   MAP_PRIVATE|MAP_ANONYMOUS,
                   -1,
                   0);
  if (rv == MAP_FAILED)
    return NULL;
  *((uint64_t *)rv) = pages;
  mprotect(rv, PAGESIZE, PROT_READ); // bug 2 fix 4096 -> PAGESIZE
  rv = (void *)((uintptr_t)rv + PAGESIZE);
  mprotect((void *)((uintptr_t)rv + (pages << PAGESHIFT)), PAGESIZE, PROT_NONE);
  return rv;
}
```

The fix for this bug is seen in the `large_alloc()` function, where the PAGESIZE was not properly allocated which causes this Floating Point exception error. The `uint64_t pages` variable should be size + PAGESIZE instead of using a fixed 4030 pagesize only. The predefined PAGESIZE definition is a much better fit for allocating the pagesizes.

Second Bug

Proof:

given input `beedbbbbX+bcqf`

the program will crash and an assert statement that checks the first malloc'ed array with the second reallocated bigger array.

```
hash: 5

array 1:
1 1 1 1 1
array 2:
0
──────────────────────── ABORT() CALLED ────────────────────────
./script1.sh: line 5:  7455 Aborted                 (core dumped) ./a.out $file
brian@DESKTOP-5USC84C:/mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4$
```

tried both instances with abort and `assert()`

given input `/`

`assert()` triggers more crashes but does not print out the second arrays content

```
String: /
buffer[i] : 0, value : 47
total: 47
hash: 5

array 1:
1 1 1 1 1
array 2:
a.out: driver.c:66: int main(int, char **): Assertion `var1[i] == var2[i]' failed.
./script1.sh: line 5:  8122 Aborted                 (core dumped) ./a.out $file
brian@DESKTOP-5USC84C:/mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4$ ./script2.sh
```

Driver Code:

```
106         case 5:
107             printf("\n---------------------------- case 5 -------------------------------\n");
108
109             var1 = sp_malloc(hash);
110             printf("array 1: \n");
111             for (int i = 0; i < hash; i++)
112             {
113                 var1[i] = 1;
114                 printf("%d", var1[i]);
115             }
116
117             printf("\n");
118
119             var2 = sp_realloc(var1, input);
120             printf("array 2: \n");
121             for (int i = 0; i < hash; i++)
122             {
123                 printf("%d", var2[i]);
124                 printf("\n-------------------- waiting for assert() call ---------------------- \n\n");
125                 if (var1[i] != var2[i]) abort();
126             }
```

Fix:

```
256  void *sp_realloc(void *chunk, size_t size) {
257      if (size == 0) {
258          sp_free(chunk);
259          return NULL;
260      }
261      uint64_t os = 0;
262      if (chunk != NULL) {
263          uintptr_t id = ((uintptr_t) chunk - ARENA_BASE) / SUBARENA_SIZE;
264          if (id < NUMCHUNKSHIFT){
265
266          // os = (1ULL << (id + MINCHUNKSHIFT)) & (!((size & 0xff)^0x37)-1);
267          // should be just
268              os = (1ULL << (id + MINCHUNKSHIFT));
269          // printf("original size : %d\n", os);
270          }
271          else
272              if (((uintptr_t)chunk & PAGEMASK) == 0ULL)
273          os = (*(uint64_t *)((uintptr_t)chunk - PAGESIZE)) << PAGESHIFT;
274              else
275          abort();
276
277      }
```

Compiling with DSP_FAKE, the abort and assert statements are not triggered. The fix for this bug lies in the code of assigning the original size of the realloc function. By printing out original size, it is always fixed to 64, assuming it is because of the extra code of : `& (!(( size & 0xff)^0x37)-1)` which causes it to be fixed to 64 all the time. This is not ideal for realloc, removing the statement solves the problem.

Other undefined bugs:

Other bugs that I could have possibly found that have no fixes to are listed below.

1. possible `sp_free()` error, when `sp_free()` is called after malloc or realloc, a segfault occurs each time it is called multiple times.

```
String: □□
buffer ASCII value : 3
buffer ASCII value : 27
hash: 6


──────────────────────────── case 6 ─────────────────────────────
AddressSanitizer:DEADLYSIGNAL
=================================================================
==28786==ERROR: AddressSanitizer: SEGV on unknown address 0x7ffffc780000 (pc 0x0000004c41fb bp 0x7fffffffede70 sp 0x7fffffffede50 T0)
==28786==The signal is caused by a READ memory access.
    #0 0x4c41fb in large_free /mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/./spalloc.c:208:20
    #1 0x4c449e in sp_free /mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/./spalloc.c:253:5
    #2 0x4c4f86 in main /mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/driver.c:125:4
    #3 0x7fffff4270b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #4 0x41b31d in _start (/mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/fuzzed_dir/default/crashes/a.out+0x41b31d)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV /mnt/c/Users/kohbr/a1782291/2021/s2/secure-programming/ass4/./spalloc.c:208:20 in large_free
==28786==ABORTING
```

```c
case 6:
    printf("\n---------------------------- case 6 ----------------------------------\n");
    var1 = sp_malloc(large_input);
    sp_free(var1);
    sp_free(var1);
    sp_free(var1);
    sp_free(var1);
    sp_free(var1);
    var1 = sp_realloc(var1, input);
    sp_free(var1);
    break;
```

2. Possible `sp_realloc()` error, calling it multiple times causes the abort statement to be triggered.

```
──────────────────────────── case 9 ─────────────────────────────
./script1.sh: line 6:  6558 Aborted                 (core dumped) ./a.out $file


String: dedbeln
buffer combined ASCII value : 718
hash value: 4
```

```c
case 9:
    printf("\n---------------------------- case 9 ----------------------------------\n");

    var1 = sp_malloc(input);
    var1 = sp_realloc(input, input);
    var1 = sp_realloc(input, input);
    var1 = sp_realloc(input, input);
    break;
```