AFL-INFER OBSERVATIONS

I initially ran AFL on Honeyd. However AFL gave me crashes which are reproducible only when the project is compiled with afl-gcc and not with any other compiler. As per Mike's suggestions, I compiled Honeyd with address sanitizer flag and still AFL gave me pseudo crashes only.

So I ran AFL on few other projects and while few projects had build issues, AFL found 0 crashes with few other. Libsodium is one such project for which AFL gave me 0 crashes. Libsodium is a software library for encryption, decryption, signatures, password hashing and more. However, Infer found multiple issues with this particular project.
Below is the link for bug report generated by Infer.
https://github.com/pguttula/AFL-infer

AFL on Libsodium:

From the test folder of libsodium,
https://github.com/TomMD/libsodium/tree/master/test/default, I tested couple of functionalities of Libsodium.

1) I initially tested a simple function *crypto_auth_hmacsha256_verify*. Original sample program can be found at - *https://github.com/TomMD/libsodium/blob/master/test/default/auth3.c*

   The sample program was modified to take input from stdin:

```
#define TEST_NAME "auth3"
#include "cmptest.h"

static unsigned char key[32] = {
   0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,0x0c, 0x0d, 0x0e,
0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16,0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
0x1d, 0x1e, 0x1f, 0x20
};

static unsigned char c[50] = { 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,0xcd, 0xcd,
0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,
0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,
```

```
0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,
0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd };

static unsigned char a[32] = { 0x37, 0x2e, 0xfc, 0xf9, 0xb4, 0x0b, 0x35, 0xc2, 0x11,
0x5b, 0x13, 0x46, 0x90, 0x3d, 0x2e, 0xf4,
0x2f, 0xce, 0xd4, 0x6f, 0x08, 0x46, 0xe7, 0x25, 0x7b, 0xb1, 0x56, 0xd3, 0xd7, 0xb3,
0x0d, 0x3f };

int
main(void)
{
  unsigned char p[10],q[10];
  int size;
  if(scanf("%9s",p) == EOF) { //Modified the highlighted parts
    return 0;                 //AFL accepts p,q,s as inputs
  }
  if(scanf("%9s",q) == EOF) {
    return 0;
  }
  if(scanf("%d",&size) == EOF){
    return 0;
  }
  printf("%d\n", crypto_auth_hmacsha256_verify(p, q, size, key));
    return 0;
}
```

After parallel fuzzing AFL on this sample file for more than a day, below are the statistics of output generated by AFL:
Crashes -0
Hangs -5
Number of paths covered-4

Similarly, I tested few other functions and there were 0 crashes found. Only difference I found was an increase in the number of hangs or the total number of paths covered by AFL.

I realized testing functions randomly did not give me any crashes/interesting results. So I took Infer bugs as reference and I fuzzed the functions with which Infer found bugs.
One of such functions is *crypto_auth_hmacsha512_init***.**
Infer Bug report on the function:

crypto_auth/hmacsha512/auth_hmacsha512.c:54: warning: ARRAY_OUT_OF_BOUNDS_L3 (biabduction/Rearrange.ml:71:54-61:)

```
 array `pad` `of size 128` could be accessed with an index out of bounds at line 54, column 9.
 52.     memset(pad, 0x36, 128);
 53.     for (i = 0; i < keylen; i++) {
 54. >       pad[i] ^= key[i];
 55.     }
 56.     crypto_hash_sha512_update(&state->ictx, pad, 128);
```

crypto_auth/hmacsha512/auth_hmacsha512.c:61: warning: ARRAY_OUT_OF_BOUNDS_L3 (biabduction/Rearrange.ml:71:54-61:)

```
 array `pad` `of size 128` could be accessed with an index out of bounds at line 61, column 9.
 59.     memset(pad, 0x5c, 128);
 60.     for (i = 0; i < keylen; i++) {
 61. >       pad[i] ^= key[i];
 62.     }
 63.     crypto_hash_sha512_update(&state->octx, pad, 128);
```

I tested AFL on the same function which takes an input of more than 128 characters. AFL found 0 crashes. Same is the case with few other bugs reported by Infer.

Conclusions

From my experiments, it is clear to me that AFL needs smarter inputs to be fed. This is because AFL explores the area around set of inputs we feed. So the input we naively feed to AFL might not lead AFL to explore all the paths in the program and might actually miss covering the path with vulnerable code in the program.

As an immediate next step, I would like to address the above problem by making use of the potential bugs discovered by infer. Given a potential bug discovered by infer, I would like to go through the code and roughly estimate what inputs might lead to the triggering of the bug. If we run AFL around these inputs, it might lead us to a crash if the bug is a real bug, and not a false positive. However, it is important to note that this requires me to understand the third party code on which infer finds bugs. These projects are very large, which means it might take me considerable time to understand the code and manually triage a bug.