AFL-INFER Observations on SQLITE

In my previous experiment, I ran both AFL & Infer on the project Libsodium. While AFL gave 0 crashes, the bugs that Infer gave me were mostly false positives. Hence I chose a version of SQLite code from 2011 as the older versions are more likely to contain real bugs and I tried to evaluate the likelihood that bugs found by infer are true positives and the effort it takes to trigger these bugs using AFL.

After running for ~10 minutes, Infer found a total of 121 issues, but most of them are warnings of dead stores and uninitialized values. Excluding these, there are 15 potential null dereferences that are of interest to us. I analyzed each such dereference, and made a triage report that can be found in the below link:

https://github.com/pguttula/InferAFL/blob/master/sqlite-version-3.6.10/issues.md

The serial numbers are the indexes of the corresponding issues in the bug report generated by infer. I have triaged bugs into three categories:
1. Bugs that are marked "verified false" are definitely false positives, mostly because the null dereference occurs after an "assert" check that infer fails to take into account.
2. Finally, bugs marked "verified unlikely" lead to crashes in unlikely events, such as malloc returning NULL due to unavailability of memory etc, which cannot be triggered through user input. Most null dereferences found by Infer fall into this category.
3. Bugs that are marked "verified true" (in bold) represent the crashes that can be triggered with a carefully determined input. There are 4 such bugs. However, as the bugs were deep inside sqlite, it is difficult to find appropriate inputs to sqlite the APIs that trigger these bugs.
Therefore, I tried to adopt a bottom-up approach to find a way to make sure that the function in which Infer reported a bug can be run on AFL. That is, instead of triggering the bugs via sqlite API, I made sqlite's internal(static) functions global, and triggered bugs in them directly.

For example, I tried to validate the below bug from Infer's report:
bld/sqlite3.c:37317 - pPage pointer returned from sqlite3PagerGetExtra can be null. Verified true. This can happen because sqlite3PagerGetExtra can explicitly return 0 when pPg->pPager is null.

This is the function that can potentially crash due to null dereference error:

```
MemPage *btreePageFromDbPage(DbPage *pDbPage, Pgno pgno, BtShared *pBt){
 Line 37317: MemPage *pPage = (MemPage*)sqlite3PagerGetExtra(pDbPage);
  pPage->aData = sqlite3PagerGetData(pDbPage);
  pPage->pDbPage = pDbPage;
  pPage->pBt = pBt;
  pPage->pgno = pgno;
  pPage->hdrOffset = pPage->pgno==1 ? 100 : 0;
  return pPage;
}
```

As we see, pPage pointer comes from sqlite3PagerGetExtra.The below function would return 0 if pPager is NULL and dereferencing it would cause an error.

```
SQLITE_PRIVATE void *sqlite3PagerGetExtra(DbPage *pPg){
  Pager *pPager = pPg->pPager;
  return (pPager?pPg->pExtra:0);
}
```

I tested this bug on AFL by making few changes to the code:

1) Modified all the functions in sqlite3.c from static to global.

2) Wrote the below program(*https://github.com/pguttula/InferAFL/blob/master/sqlite-version-3.6.10/bld/afl-test.c*) to access the function **btreePageFromDbPage** and ran it on AFL.

```
#include<stdio.h>
#include "sqlite3.c"
typedef u32 Pgno;
extern MemPage *btreePageFromDbPage(DbPage *pDbPage, Pgno pgno, BtShared *pBt);

int main() {

  Pgno pgno;
  uint64_t ptr1,ptr2;

  if(scanf("%32u",&pgno) <= 0) {
    exit(0);
  }
  if(scanf("%lu",&ptr2) <= 0){
    exit(0);
  }
  BtShared* pBt;
  pBt=(BtShared*)ptr2;
  if(scanf("%lu",&ptr1) <= 0) {
    exit(0);
  }
  DbPage* pDbPage;
  if(ptr1 == 22453) {
    pDbPage = (DbPage*)malloc(sizeof(struct PgHdr));
    pDbPage->pPager = (Pager *)malloc(sizeof(struct Pager));
    pDbPage->pExtra = malloc(sizeof(struct MemPage));
  }
  else {
    pDbPage = (DbPage*)malloc(sizeof(struct PgHdr));
    pDbPage->pPager = NULL;
    //pDbPage->pPager = (Pager *)malloc(sizeof(struct Pager));
    pDbPage->pExtra = malloc(sizeof(struct MemPage));
  }
//  printf("here");
  btreePageFromDbPage(pDbPage, pgno,pBt);
//  printf("%d",i);
  return 0;
}
```

From the above code, if ptr1 is anything other than 22453, the program crashes as pPager in the other would be null and hence there would be a NULL dereference error. AFL took 6 seconds to generate this crash(*https://github.com/pguttula/InferAFL/tree/master/sqlite-version-3.6.10/bld/.libs/sync_dir/fuzzer01/hangs*). The other 3 bugs which were verified true can be generated using AFL in the same manner.