0-DAY NO CVE HIGH PUBLIC

SQLite 3.53.4 FULL JOIN USING skips sqlite3AuthRead

SQLite 3.53.4 lookupName rewrites FULL JOIN USING to coalesce() TK_FUNCTION. AuthRead runs only for TK_COLUMN/TK_TRIGGER, so a READ-denied column comes back. Authorizer is off on default open. Labbed 3.53.4. No CVE yet.

NAME SQLite 3.53.4 FULL JOIN USING skips sqlite3AuthRead	TYPE 0-day analysis
CVE n/a	CVE RISK high
DISCLOSURE STATUS public	VENDOR SQLite (Hwaci)
AFFECTED SQLite 3.53.4 (b09c88c, SOURCE_ID 2026-07-24 19:02:57 bf7c7f3). Embedder sqlite3_set_authorizer. Authorizer off on default open. 3.54-only fix.	PUBLISHED 09 Oct 2026
UPDATED 09 Oct 2026	TAGS 0-day, sqlite, authorizer, privilege-leftover, cwe-863, cwe-284, local

TL;DR

Lab files and proof-of-concept here.

01 SELECT x FROM t2 is prohibited. FULL JOIN USING hands it back.

I am @abraxas_null. The proof of concept is on GitHub: **abraxas/sqlite-fj-using-auth** (loopback client). The lab stack is `lab/`: Dockerfile, docker-compose.yml, run.sh, poc.py, auth.c. Authorized lab only. It talks to loopback.

This is **SQLite 3.53.4** (`b09c88c14082339b66c7b7158d609a771e64ca69`). SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`. Hwaci. GitHub is a Fossil mirror. Public domain engine, AGPLv3 pack. No CVE yet. CWE-863 / CWE-284. 7.5 High (`CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N`). Local. Privilege leftover of `sqlite3_set_authorizer`.

The callback is **off on default open**. `sqlite3_set_authorizer` stores NULL until the embedder installs one. This leftover is of that C API, not of `sqlite3_open` with no authorizer. Not RCE. Not a crash. Not a WAL disclosure. A column the authorizer DENY/IGNORE'd on `SQLITE_READ` still comes back through FULL JOIN USING.

02 After PHP, the library everyone embeds

I had just finished an unpublished Crit/High review of PHP 8.5.11. PHP's security model is a language runtime. SQLite's is a C library that sits inside PHP, inside browsers, inside mobile apps, inside every "we shipped a .db" product that did not want to run a server. sqlite.org/security.html is unusually honest about the input: untrusted SQL, and an untrusted database file, are first-class. A leftover that only fires when the host installed an authorizer is still a leftover of a documented security boundary. The header file says so.

I only wanted Critical or High. Local is in-scope. Medium stays on the floor unless it chains. A finding on this hunt needed memory corruption or crash on untrusted SQL

or an untrusted `.db`, an info leak from a crafted WAL, a leftover of a named 3.54 check-in that still fires on latest GA, or a documented C API that claims to deny a column and then returns it.

Default `load_extension` is off. I am not filing "the host turned extensions on." Default `TRUSTED_SCHEMA` is ON for compat. That default itself is not a bug.

03 Pinning 3.53.4, not the draft

Latest GA. Not an RC. Not 3.54.0. The 3.54 branch was still draft when I pinned, planned around 2026-10-15. Pinning a draft is how you "discover" a bug the next tag already closed. The Fossil release is 2026-07-24. Git tag `version-3.53.4` is `b09c88c14082339b66c7b7158d609a771e64ca69`. Runtime `sqlite3_sourceid()` is `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`. The lab refuses to proceed if either string is wrong.

3.53.4 is the AI-bug maintenance wave on 3.53.0-3.53.3. If a leftover of a later trunk check-in still fires here, it is not a stale minor. Canonical tree is sqlite.org/src. GitHub is the mirror I can deep-link.

04 The skip list is half the hunt

Before you look for leftovers, you write down what is already public so you do not "discover" a CVE with a press release. On 3.53.4 the interesting ones were already closed:

- CVE-2026-11822 / CVE-2026-11824 FTS5 heap overflow when DEFENSIVE is off. Fixed 3.53.2. `fts5LeafRead` rejects `szLeaf<4`. In tree.
- Ada Logics `sqlite3_rsync` stack smash. `readPow2` `x>16`. In tree. The later SSH filename leftover is a different finding, a different pack.

- CVE-2026-50812 session `changeset_apply_v3` NPD. Medium. Fixed 3.53.2. Optional `-DSQLITE_ENABLE_SESSION`.
- CVE-2025-3277 / CVE-2025-29087 `concat_ws` integer overflow. Fixed 3.49.1.
- CVE-2025-6965 integer overflow read. Fixed 3.50.2.
- CVE-2025-7709 FTS5 corrupt-index overflow. Fixed 3.50.3.
- CVE-2025-29088 / CVE-2025-52099 LOOKASIDE OOB. Fixed 3.49.1. `setupLookaside` i64 clamp is in tree.
- CVE-2025-70873 zipfile CLI OOB. Fixed 3.52.0. CLI extra.
- CVE-2024-0232 JSON UAF. Fixed 3.43.2. The JSON subtype/UPLUS walk is the Fossil parent of this tag.
- CVE-2023-7104 session on a corrupted changeset. Fixed 3.43.1.
- CVE-2022-35737 C-API array overflow on huge strings. Not reachable from SQL or a DB file.
- WAL-reset (Tailscale race). Fixed 3.51.3 / 3.53.0.
- CVE-2026-51296 through 51304. Unreproducible. SQLite and JFrog treat them as AI hallucinations. I am not re-filing a slop CVE.

The skip list also named the leftovers of later trunk that had no CVE and were not in this GA. FULL JOIN USING authorizer was one of them. Forum 2026-07-24T02:52:08Z and 2026-05-31T02:00:37Z (yuelinwang). Trunk check-in the same calendar day as the GA, trunk only, "will not appear in an official release until 3.54.0." That sentence is why this pin still has the skip.

05 Twenty surfaces, two kinds of leftover

I split the tree. Parser / VDBE. btree / pager. WAL. FTS3. FTS5. JSON. vtab / ATTACH / pragma. trusted_schema / authorizer. shell / zipfile. sqlite3_rsync. session / RBU. deserialize / memdb. triggers / views / window. rtree / geopoly. WASM. 3.54 patch siblings. load_extension / VFS. C API. recover / misc extras. schema side effects.

Most of that is SQLite doing what 3.53.4 said it would do.

`load_extension` SQL and C API deny unless the host sets the enable bits. Function is DIRECTONLY|UNSAFE. FTS3 NEAR negative-delta overwrite is in this pin. JSON UAF class is closed. LOOKASIDE is clamped. zipfile remaining OOB of 70873 is closed. Default ATTACH / VACUUM INTO any writable path is documented. `sqlite_log()` from a hostile schema is log injection only (Medium; declined backport). `vec1_train` is not in this tree. JS kvvfs overwrite is official WASM extra, not default native. overflow-page encoding and LEFT JOIN IN are correctness, not shown memory-safety High.

Default core unpublished Crit/High is not empty. COLLATE depth, CTE nest, and VALUES UNION ALL still blow the C stack on untrusted SQL. A crafted WAL whose header page size is a valid power of two smaller than the database page size still short-reads into a full pager buffer with no tail memset. Those are other posts. This one is the authorizer.

Authorizer embedders still skip `SQLITE_READ` on FULL JOIN USING, and on `INSERT INTO dest SELECT * FROM src` (xferOptimization). The second is a sibling leftover. DRH treated the 3.54 xfer check-in as a new feature, not a branch-3.53 bugfix. I am not bundling them. This pack is the JOIN.

06 The API the docs sold as column ACL

`sqlite.h.in` is unusually plain:

> The authorizer is disabled by default.

And, on `SQLITE_READ` plus `SQLITE_IGNORE`:

> The SQLITE_IGNORE return can be used to deny an untrusted user access to individual columns of a table.

And:

> An authorizer is used when preparing SQL statements from an untrusted source, to ensure that the SQL statements do not try to access data they are not allowed to see.

That is the victim. Chrome/WebSQL-style hosts. PHP `SQLite3::setAuthorizer`. Any embedder that compiles untrusted SQL under a callback and treats DENY/IGNORE on `SQLITE_READ` as the column gate. Authorization is compiled in. Default Makefiles do not set `SQLITE_OMIT_AUTHORIZATION`. The callback pointer is still NULL until someone calls `sqlite3_set_authorizer`.

`sqlite3AuthRead` / `sqlite3AuthReadCol` (`auth.c`) implement IGNORE by rewriting the node to `TK_NULL` and DENY by `access to %z is prohibited` / `SQLITE_AUTH`. Those paths have to run. If name resolution turns the column into a function before the gate, they never run.

`test/auth.test` on this pin has no FULL JOIN case. Tests that do not exist do not catch rewrites that did not exist when the authorizer was written.

07 The rewrite that changes the opcode

FULL JOIN USING (and NATURAL FULL JOIN, which is the same USING machinery) cannot pick a single table for a shared column. INNER and LEFT USING take the left-most table and `continue`. RIGHT JOIN clears the match list. FULL JOIN has to return the non-NULL side. `lookupName()` builds that as a synthetic `coalesce()`:

```c
if( pFJMatch ){
  if( pFJMatch->nExpr==cnt-1 ){
    extendFJMatch(pParse, &pFJMatch, pMatch, pExpr->iColumn);
    pExpr->op = TK_FUNCTION;
    pExpr->u.zToken = "coalesce";
    pExpr->x.pList = pFJMatch;
    pExpr->affExpr = SQLITE_AFF_DEFER;
    cnt = 1;
    goto lookupname_end;
```

`resolve.c:763-779`. The parent is no longer a column. It is a function named `coalesce`. Then it jumps.

The arguments are built by `extendFJMatch()` as raw `TK_COLUMN` nodes. Cursor, column number, table pointer. No `sqlite3AuthRead`. No walk through `lookupName` for those children:

```c
static void extendFJMatch(...){
  Expr *pNew = sqlite3ExprAlloc(pParse->db, TK_COLUMN, 0, 0);
  if( pNew ){
    pNew->iTable = pMatch->iCursor;
    pNew->iColumn = iColumn;
    pNew->y.pTab = pMatch->pSTab;
    *ppList = sqlite3ExprListAppend(pParse, *ppList, pNew);
  }
}
```

`resolve.c:208-223`.

The only authorizer site in `lookupName()` is after that jump, and it keys off the **parent** opcode:

```c
lookupname_end:
  if( cnt==1 ){
#ifndef SQLITE_OMIT_AUTHORIZATION
    if( pParse->db->xAuth
     && (pExpr->op==TK_COLUMN || pExpr->op==TK_TRIGGER)
    ){
      sqlite3AuthRead(pParse, pExpr, pSchema, pNC->pSrcList);
    }
#endif
    ...
    return WRC_Prune;
```

`resolve.c:836-854`. On this pin `pExpr->op` is `TK_FUNCTION`, so `sqlite3AuthRead` is **never** called for any USING term, including the head table. `WRC_Prune` also stops the walker from descending into `pFJMatch->a[0..n]`. The coalesce arguments are live columns with no READ callback.

That is strictly worse than the first trunk attempt (`bbd69ce274`), which authorized `a[0]` `nExpr` times and still skipped `a[1..]`. The working trunk fix is `2cb2287768` / Git `5694c3bb` (2026-07-24 14:53, same calendar day as the GA). It loops AuthRead over every `pFJMatch` child. Trunk only. 3.54.0. "Obscure and harmless." Not back-ported to 3.53.x.

LEFT JOIN USING does not take this rewrite. FULL JOIN ON does not take this rewrite. Explicit `coalesce(NULL, t2.x)` still walks `t2.x` as `TK_COLUMN` and still authorizes. The leftover is the USING rewrite, not coalesce as a function.

08 The lab

I did not want a Python sqlite3 binding in the middle of the proof. The C API is the victim. The harness is `lab/auth.c`: open `:memory:`, create `t1(x,y)` and

`t2(x,y2)`, put the witness in `t2.x`, install an authorizer that DENY or IGNORE `READ t2.x` and logs every callback, then prepare four shapes.

The image is not a Hub SQLite tag. Official `sqlite-src-3530400.zip` from sqlite.org, amalgamation, `cc` of `auth.c` plus `sqlite3.c`. Runtime checks `sqlite3_libversion() == 3.53.4` and the SOURCE_ID token. If you swap the zip, the binary refuses.

Compose project `sqlite-fj-using-auth`. Loopback `127.0.0.1:18840`. The published port is a tiny ready HTTP server that prints the witness so `poc.py` knows the container is up. The proof is `docker compose exec` of `/usr/local/bin/fj-using-auth`, not SQL over HTTP.

DENY, **`SELECT x FROM t2`**. Prepare `SQLITE_AUTH` 23. `access to t2.x is prohibited`. One `READ t2 x`. No witness. That is the control the docs described.

DENY, **`SELECT x FROM t1 FULL JOIN t2 USING(x)`**. Prepare OK. Two rows: `1` and `SQLITE-FJ-USING-AUTH-WITNESS`. `read_t2x=0`. Authorizer log is `SELECT` and nothing else. The denied column came back. Zero READ callbacks on it.

DENY, **`SELECT t2.x FROM t1 FULL JOIN t2 ON t1.x = t2.x`**. Prepare 23. Same prohibited message. One `READ t2 x`. ON is not USING. The leftover is the rewrite, not FULL JOIN as a join type.

DENY, **`SELECT x FROM t1 LEFT JOIN t2 USING(x)`**. Prepare OK. One row, `1`. No witness. One `READ t1 x`. Left-most table, no coalesce rewrite.

DENY, **`SELECT coalesce(NULL, t2.x) FROM t2`**. Prepare 23. Function callback plus `READ t2 x`. Explicit coalesce still authorizes the column argument.

IGNORE, **`SELECT x FROM t2`**. Prepare OK. One row, NULL. The IGNORE rewrite to `TK_NULL` still runs when the parent is a column.

IGNORE, FULL JOIN USING. Prepare OK. Witness comes back. IGNORE never ran on `t2.x` because AuthRead never ran.

Last line:

```
SUCCESS sqlite-fj-using-auth how=full-join-using-skip-authread baseline=denied fj
```

Replay:

```
cd lab && ./run.sh
```

Or from a fresh clone:

```
git clone https://github.com/abraxas/sqlite-fj-using-auth
cd sqlite-fj-using-auth/lab
./run.sh
```

Bind it to loopback. The loopback client is `lab/run.sh` with a banner on it.

09 Wrong turns, in the order I made them

Treating authorizer as default-on. The header is explicit. NULL callback. Default `sqlite3_open` is not this leftover. Scoring it as a default-library RCE is how you write a bad advisory. Scoring it as "not a bug because default is off" is how you ignore the API the same header sold as column ACL. Leftover of that C API. High against embedders that installed it.

Pinning 3.54.0 while it was still draft. The fix landed on trunk the same day as 3.53.4. If you pin trunk you "prove" the bug is gone and you have labbed a branch nobody ships. Latest GA is the pin. Draft is the skip.

Re-filing the forum item as if 3.53.4 had the loop. The skip list already named `2cb2287768`. Confirming it is absent from `b09c88c` is the hunt. The lab is the confirmation. I am not claiming first-finder on the JOIN shape. I am claiming latest GA still returns the column.

Expecting LEFT JOIN USING to leak. INNER/LEFT USING `continue`s on the left-most table. `t1.x` is authorized, `t2.x` is never read. The row is `1`. No witness. That is a negative control, not a miss.

Expecting FULL JOIN ON to leak. ON is an ordinary expression. `t2.x` stays `TK_COLUMN`. AuthRead fires. Prepare 23. If your PoC only has USING, you have not shown the rewrite is the sink.

Expecting explicit `coalesce(t2.x)` to skip AuthRead. User-written coalesce still walks arguments as columns. The leftover is the **synthetic** coalesce that `lookupName` builds and then `goto lookupname_end`s past the gate. The lab includes that query so nobody can pretend "coalesce is the bug."

Treating the incomplete trunk fix as the fix. `bbd69ce274` authorized `a[0]` repeatedly. The denied table is often `a[1]`. Loop every child or it is not a fix. 3.54 has the loop. 3.53.4 has neither.

Bundling xferOptimization. `INSERT INTO dest SELECT * FROM src` skipping source READ is the same authorizer boundary and a different function (`xferOptimization` in `insert.c`). Different lab, different pack. DRH called that one a 3.54 feature. This JOIN leftover was called obscure and harmless. Both still skip real column data on this GA.

Labbing with Python's sqlite3 module and calling it the C API. The stdlib binding is a wrapper. The victim is `sqlite3_prepare_v2` after `sqlite3_set_authorizer`. `auth.c` against the amalgamation is the oracle. Python is the compose waiter.

Calling default open this bug. Authorizer off. Listed as leftover of that API. The SUCCESS line is against an installed DENY/IGNORE callback. I am not going to pretend every `sqlite3_open` in the wild grew a column ACL.

A reverse shell. Theatre. The oracle is a witness string from a DENY'd column on loopback.

Inflating this to RCE. SQLite does not eval the column. The leftover is read of a column the callback refused. Confidentiality of that column, C:H I:N A:N, local, PR:L because the attacker already prepares SQL against a connection that has an authorizer. Not code execution in the host.

10 What an attacker can do

Against an embedder that installed `sqlite3_set_authorizer` as column ACL, prepare SQL. `SELECT x FROM t2` is denied. `SELECT x FROM t1 FULL JOIN t2 USING(x)` (or NATURAL FULL JOIN) returns `t2.x`. Zero `READ t2.x` callbacks.

The authorizer is off on default open. This is not a leftover of stock `sqlite3_open`. It is a leftover of the documented C API that exists so hosts can compile untrusted SQL and hide columns. Hosts that never call `sqlite3_set_authorizer` are not this victim. Hosts that do, and then trust DENY/IGNORE on `SQLITE_READ`, still hand the column back through FULL JOIN USING.

Not RCE. Not a crash. Not a file write. Privilege leftover. Local. The attacker already talks to that connection. The connection already has a callback. The callback already said no. The JOIN said yes.

I am not going to print a snippet you can paste into someone else's app. The missing AuthRead on `TK_FUNCTION` after the USING rewrite is the useful part. If you own the embedder, run the proof of concept against loopback.

11 The fix

Loop `sqlite3AuthRead` over every `pFJMatch` child after the coalesce rewrite, the way trunk `2cb2287768` already does for 3.54. Authorizing only `a[0]` is not enough. Do not `WRC_Prune` past those children until they have been through AuthRead.

Until a 3.53 backport or a 3.54 GA, embedders that use the authorizer as column ACL should reject FULL JOIN / NATURAL JOIN from untrusted SQL, or deny `SQLITE_SELECT` on the protected table entirely so the statement never compiles. Column-level DENY on USING columns is not a control on this pin.

Do not take "authorizer is off by default" as a reason to skip the gate in `lookupName`. The gate exists because the callback exists. HALF a gate is how leftovers look when they ship.

12 References

- Proof of concept: abraxas/sqlite-fj-using-auth · sqlite-fj-using-auth-Abraxas-Labs.py
- Lab: `lab/` · Dockerfile · docker-compose.yml · run.sh · poc.py · auth.c
- @abraxas_null · github.com/abraxas · abraxaslabs.tech · abraxas.null@proton.me
- CWE-863 · CWE-284 · 7.5 High
- sqlite.org/src tag version-3.53.4 (`b09c88c14082339b66c7b7158d609a771e64ca69`)
- `src/resolve.c` `lookupName` / `extendFJMatch`
- `src/auth.c` `sqlite3AuthRead` / `sqlite3AuthReadCol`
- Forum 2026-07-24T02:52:08Z · 2026-05-31T02:00:37Z



Abraxas Labs

Independent security research by abraxas.

Analysis of 0-days, n-days, malware, and the

leftovers that still matter.

GitHub **@abraxas_null** **abraxas.null@proton.me** **abraxaslabs.tech**

Home **Research** **Open Source** **Reading Room** **About** **GitHub** **@abraxas_null** **Console**

Independent research. Private analysis by arrangement. No ads. abraxas.null@proton.me · For defensive awareness and authorized testing only.

© 2026 Abraxas Labs