0-DAY NO CVE HIGH PUBLIC

SQLite nested CTEs skip the view-depth cap

SQLite 3.53.4 increments nNestSel for nested VIEWs and errors too-deep. CTE attach does not. A 25000-deep WITH chain stack-overflows selectExpander. Crash, not RCE. No CVE yet.

NAME SQLite nested CTEs skip the view-depth cap	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). Leftover of later trunk CTE nNestSel. Nested VIEWs already capped on this pin. 3.54.0 is draft only.	PUBLISHED 09 Oct 2026
UPDATED 09 Oct 2026	TAGS 0-day, sqlite, cte, cwe-674, crash, stack-overflow

TL;DR

Lab files and proof-of-concept here.

01 The cap that only counts VIEWs

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

This is **SQLite 3.53.4** (`b09c88c14082339b66c7b7158d609a771e64ca69` , SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`). Hwaci. Public domain engine, AGPLv3 pack. No CVE yet. CWE-674. 7.5 High (`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H`). Reach is **Local**: untrusted SQL or a nested CTE schema. Class is **Crash**. Not RCE.

I built twenty nested VIEWs, set `.limit expr_depth 20` , and SQLite said `VIEWs and/or subqueries nested too deep` . That is the documented defense. `Parse.nNestSel` ticks in `sqlite3ResultSetOfSelect` . At `SQLITE_LIMIT_EXPR_DEPTH` the prepare fails and the process lives.

Then I wrote the same shape as CTEs. `WITH c0 AS (SELECT 1), c1 AS (SELECT * FROM c0), ...` out to twenty-five thousand names. The view-depth counter did not move. `selectExpander` walked each CTE as a nested FROM subquery. `sqlite3WalkSelect` called itself until AddressSanitizer printed `stack-overflow` in the amalgamation and the CLI died. Nested FROM at the same N was a parser `Recursion limit` . Nested VIEWs were too-deep. Only the CTE chain crashed.

A cap that counts VIEWs and forgets CTEs is not a cap. It is a comment that got compiled into one path.

02 Why SQLite

I had just finished unpublished Crit/High hunts of Caddy, lighttpd, PHP, and OpenLiteSpeed. Web servers and a CGI leftover. SQLite is the other thing those

stacks sit on: a C library that will prepare whatever SQL you hand it, including a hostile `.db` whose schema is the payload.

sqlite.org/security.html is unusually honest for a database. Untrusted SQL is a first-class input. Untrusted database files are a first-class input. A crash or a heap disclosure on those is a serious bug. `load_extension` is off on default open. `TRUSTED_SCHEMA` is ON because twenty years of applications stored VIEWs and expected them to run. The documented depth limit is supposed to keep the C stack from becoming the query planner.

I only wanted Critical or High. Local is in-scope. Medium log injection stays on the floor. Superuser `load_extension` after the app turns the bit on is intended. ATTACH to any writable path is documented. I was not going to file "SQLite will execute SQL."

The product people actually ship is the amalgamation. GitHub is a Fossil mirror. Canonical is sqlite.org/src. Developers fix in hours and often do not write CVEs. That is not a reason to skip the leftover. It is a reason not to open a public issue on sqlite/sqlite.

03 Pinning 3.53.4, not the draft

Latest GA. Tag `version-3.53.4`. Git `b09c88c14082339b66c7b7158d609a771e64ca69`. Fossil release 2026-07-24. `SELECT sqlite_source_id()` returns `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`. That string is the pin. The lab greps it after the zip SHA3.

3.54.0 was still a draft when I pinned (planned 2026-10-15). Drafts move. GA does not, until the next one. Pinning a draft is how you "find" a bug the next tarball already closed, or miss one the draft has not landed yet. I did not pin 3.54.0.

The official source zip is `sqlite-src-3530400.zip` from sqlite.org/2026/. SHA3-256 `b834d474b9b393d85a9e3ee4cc11f1329e007e9376a424ee740796f5c4bda3a8`. The Dockerfile checks that hash before `configure`. ASan CFLAGS, 8 MiB stack ulimit, loopback `127.0.0.1:18810`.

3.53.4 itself is a maintenance patch for (mostly) AI-discovered bugs in 3.53.0-3.53.3. That wave is closed on this pin. The leftovers I wanted were later trunk versus this GA.

04 The skip list is half the hunt

sqlite.org/cves.html is the public table. On 3.53.4 the interesting ones were already closed. I wrote them down so I would not "discover" a CVE with a check-in.

- CVE-2026-11822 / CVE-2026-11824. FTS5 heap overflow when `SQLITE_DBCONFIG_DEFENSIVE` is off. Duplicate CVEs for one bug. Fixed 3.53.2. `fts5LeafRead` already has `szLeaf<4`. I still read FTS5. The leftover there is a different function.
- CVE-2026-50812. Session `sqlite3changeset_apply_v3` NULL deref on a malformed changeset. Medium. Fixed 3.53.2. Optional `-DSQLITE_ENABLE_SESSION`.
- CVE-2025-3277 / CVE-2025-29087. `concat_ws()` integer overflow / heap write. Fixed 3.49.1.
- CVE-2025-29088 / CVE-2025-52099. `sqlite3_db_config` LOOKASIDE OOB. Fixed 3.49.1. `setupLookaside` already clamps.
- CVE-2025-70873. zipfile CLI OOB on a malformed ZIP. Fixed 3.52.0. CLI extra, not default lib.
- CVE-2024-0232. JSON UAF from injected SQL. Fixed 3.43.2. The subtype/UPLUS walk is the Fossil parent of this tag.
- Ada Logics `sqlite3_rsync` stack smash. `readPow2` `x>16` is in tree (`805f3acd`, 2026-07-09, before this GA). ORIGIN_PAGE no longer writes a

64KiB stack buffer from an attacker page size. The rsync leftover that remains is SSH filename quoting, which is a different pack.

Also skipped, because they are not bugs in this library: CVE-2026-51296 through 51304 (JFrog: AI hallucinations), SQLite JDBC, LuxCal, Node bindings, `.expert`, `appendvfs`.

`load_extension` SQL and C API still deny unless the embedder sets the bits. Function is `DIRECTONLY|UNSAFE`. I did not spend a week proving that off is off.

05 Twenty hunts

I split the tree. Parser and VDBE. Btree and pager. WAL. FTS3. FTS5. JSON. Virtual tables, ATTACH, pragma. `trusted_schema` and the authorizer. Shell and zipfile. `sqlite3_rsync`. Session and RBU. Deserialize and memdb. Triggers, views, window functions. rtree and geopoly. WASM. 3.54 patch siblings. `load_extension` and VFS. The C API. Recover and the misc extras. Schema side effects.

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

JSON is not the 2024 UAF. WAL-reset corruption is closed. zipfile remaining OOB of CVE-2025-70873 is closed. Default ATTACH is documented. `sqlite_log()` from a hostile schema is log injection; DRH declined a 3.53 backport. JS kvvfs is WASM extra. Public forum btree recursion crashes are already on sqlite.org/bugs.

The parser hunt is where this leftover lives. COLLATE depth is a sibling in the same file: postfix `COLLATE BINARY` never calls `sqlite3ExprCheckHeight()`, so `SQLITE_LIMIT_EXPR_DEPTH` never runs on that operator. This post is not that bug. This post is the CTE attach path that does not tick `nNestSel`.

Triggers and views as a surface came back NONE for unpublished schema RCE. Window frames are range-checked. INSTEAD OF rewrites to BEFORE. The view-depth cap that fired in my lab is the 2026-06-16 nested-VIEW work (`3f3fb9b638` /

branch-3.53 `38430aa5ed`). It is in this GA. It is also the control that makes the CTE miss interesting: the same process, the same limit, two syntaxes, one crash.

06 The leftover

`Parse.nNestSel` has one increment on this pin. `sqlite3ResultSetOfSelect` in `src/select.c`:

```c
pParse->nNestSel++;
#if SQLITE_MAX_EXPR_DEPTH>0
  if( pParse->nNestSel >= db->aLimit[SQLITE_LIMIT_EXPR_DEPTH] ){
    sqlite3ErrorMsg(pParse, "VIEWs and/or subqueries nested too deep");
    return 0;
  }
#endif
```

That function builds a `Table` that describes a SELECT result set. VIEW column-name resolution goes through it. Nested VIEWs therefore hit `SQLITE_LIMIT_EXPR_DEPTH` (default 1000, or whatever `.limit expr_depth` set).

CTE attach does not go through it. Around `select.c:5754` the WITH item becomes a subquery:

```c
pFrom->pSTab = pTab;
/* ... */
sqlite3SrcItemAttachSubquery(pParse, pFrom, pCte->pSelect, 1);
```

No `nNestSel++`. No too-deep error. The CTE body is now `pFrom->u4.pSubq->pSelect` with `SF_CopyCte`.

The walker is recursive. `sqlite3WalkSelectFrom` in `src/walker.c`:

```c
if( pItem->fg.isSubquery
 && sqlite3WalkSelect(pWalker, pItem->u4.pSubq->pSelect)
){
  return WRC_Abort;
}
```

`sqlite3SelectExpand` installs `selectExpander` as the select callback and calls `sqlite3WalkSelect`. A chain of N CTEs is N nested FROM subqueries. Each expander frame calls the walker, which calls the expander. C stack, not SQLite heap. `SQLITE_LIMIT_EXPR_DEPTH` is an integer on `Parse`. It does not watch native frames.

Trunk `f547308713` (2026-10-05) is titled "Limit the depth of recursion for CTEs using SQLITE_LIMIT_EXPR_DEPTH". Forum bug `2026-10-04T04:58:58Z`. After this GA. No CVE on the public table when I pinned.

A chain of N=5 is a row. That is not a fix. That is the happy path the tests already cover.

07 Lab

Official source zip, ASan amalgamation, compose project `sqlite-cte-nest`, published `127.0.0.1:18810`. Ready HTTP returns `ok SQLITE-CTE-NEST-WITNESS`. Pin queries must print `3.53.4` and the full SOURCE_ID or the PoC fails closed.

Replay:

```
git clone https://github.com/abraxas/sqlite-cte-nest
cd sqlite-cte-nest/lab
./run.sh
```

Bind it to loopback. The loopback client is `lab/run.sh` with a banner on it. Do not point this at anyone else's SQLite.

The image is not a network SQLite server. The library has no listener. The 8080 inside the container is a ready probe so the PoC can wait without parsing compose logs. SQL runs as `sqlite3 :memory: < /work/....sql` with `ASAN_OPTIONS=detect_leaks=0:abort_on_error=1:halt_on_error=1:print_stacktrace=1`. Stack ulimit 8388608.

08 Wrong turns, in the order I made them

Pinning 3.54.0 because the calendar said October. It was a draft. The leftover is "later trunk versus latest GA." Draft is later trunk. Pin the tarball people compile.

Treating nested VIEWs as the crash. I wrote `CREATE VIEW v0 AS SELECT 1` then `v1` from `v0` out to depth 25, then `.limit expr_depth 20` and `SELECT * FROM v25`. Parse error: `VIEWs and/or subqueries nested too deep`. Return code 1. No ASan. That is the control. If nested VIEWs crashed, the 2026-06-16 cap would be missing and this would be a different bug. The PoC fails if that path overflows.

Treating nested FROM as the crash. `SELECT * FROM (` repeated 25000 times around `SELECT 1`. Parser `Recursion limit`. No `selectExpander` stack death. Lemon and the tokenizer have their own recursion budget. The leftover is not "SQL can be deep." The leftover is "WITH attach skips the counter the VIEW path uses."

Taking N=5 as proof the cap applies. Five CTEs return `1`. Of course they do. The view-depth default is 1000. A passing small chain is the negative control, not the oracle. The PoC requires `small-cte=ok` before it is allowed to crash.

Expecting the parser to refuse the WITH list. A 25000-name WITH is a long token stream, not 25000 nested parens. `SQLITE_LIMIT_SQL_LENGTH` defaults to a billion bytes. The statement fits. The C stack does not.

Looking for RIP control. ASan `stack-overflow` in `selectExpander` / `sqlite3WalkSelect` is a crash. The frames are predictable. Predictable frames on a

stack overflow are not instruction-pointer control in this pack. I did not write a ROP chain. The class is Crash.

Running without ASan and calling a silent abort "unclear." Without the sanitizer you still die. With it you get `#11 ... selectExpander ... sqlite3.c:155030` and `#12 ... sqlite3WalkSelect ... sqlite3.c:110430` repeating until SUMMARY. That is the witness I wanted.

Putting the crash SQL in the Dockerfile. The image must boot. The ready probe must answer. The PoC writes `/work/crash.sql` at runtime. A build that dies in `configure` is not a lab.

Calling this remote because CVSS says AV:N. SQLite's security model treats injected SQL as a local input to the host process. The attacker already has a query or a `.db`. They do not get a new network listener. Reach in the table is Local. The 7.5 is availability of that process.

09 What an attacker can do

Get SQL into the victim process, or get the victim to open a `.db` whose schema holds the WITH chain.

PLAIN TEXT Copy

```
WITH c0 AS (SELECT 1),
     c1 AS (SELECT * FROM c0),
     c2 AS (SELECT * FROM c1)
     -- ... c25000 ...
SELECT * FROM c25000;
```

`selectExpander` recurses until the host crashes. Nested VIEWs of the same idea are already capped. Nested FROM is rejected. This syntax is not.

Typical victims are the ones SQLite already listed: applications that prepare untrusted SQL, or that open a database file from a user, a sync folder, a backup, a content-

disposition, a document. The crash is the host UID. SQLite does not eval CTE names as shell. There is no `system()` on this path. Not RCE.

On the lab amalgamation, N=25000 is enough for ASan stack-overflow at an 8 MiB ulimit. Smaller N survived. Larger N was not required. The SUCCESS line records the N that died.

10 The fix

Increment `nNestSel` on CTE attach the way `sqlite3ResultSetOfSelect` already does for VIEWs. Error at `SQLITE_LIMIT_EXPR_DEPTH` with the same too-deep message, or a CTE-specific one. Decrement on the way out so a sequence of shallow WITHs still works.

Trunk `f547308713` (2026-10-05) is that class of patch. It is not in 3.53.4. Until it is backported or 3.54.0 is actually released, a host that prepares untrusted SQL on this GA can be crashed with a WITH chain.

I am not going to print a 25000-line WITH you can paste at someone else's `sqlite3`. The missing increment is the useful part. If you own the process, run the proof of concept against loopback.

PLAIN TEXT Copy

```
SUCCESS sqlite-cte-nest how=asan-stack-overflow crash-n=25000 small-cte=ok nested
```

11 References

- Proof of concept: abraxas/sqlite-cte-nest · sqlite-cte-nest-Abraxas-Labs.py
- Lab: `lab/` · Dockerfile · docker-compose.yml · run.sh · poc.py

- @abraxas_null · github.com/abraxas · abraxaslabs.tech · abraxas.null@proton.me
- CWE-674 · 7.5 High `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H`
- Tree: sqlite/sqlite version-3.53.4 (`b09c88c14082339b66c7b7158d609a771e64ca69`) · canonical sqlite.org/src · SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`
- `src/select.c` `sqlite3ResultSetOfSelect` (`nNestSel++`) · CTE attach `:5754` `sqlite3SrcItemAttachSubquery` · `selectExpander` / `sqlite3SelectExpand`
- `src/walker.c` `sqlite3WalkSelectFrom`
- Forum bug `2026-10-04T04:58:58Z` · trunk `f547308713` (2026-10-05)
- Nested VIEW cap on this pin: `3f3fb9b638` / branch-3.53 `38430aa5ed` (2026-06-16)
- CVE table: sqlite.org/cves.html · security model: sqlite.org/security.html

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**

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

© 2026 Abraxas Labs