0-DAY NO CVE HIGH PUBLIC

SQLite COLLATE chain skips the expression-depth limit

SQLite 3.53.4 COLLATE postfix never calls sqlite3ExprCheckHeight. A 120k COLLATE chain ASan stack-overflows in sqlite3WalkExprNN. Depth limit 1000 is skipped. Labbed. No CVE yet.

NAME SQLite COLLATE chain skips the expression-depth limit	TYPE 0-day analysis
CVE n/a	CVE RISK high
DISCLOSURE STATUS public	VENDOR SQLite (Hwaci)
AFFECTED SQLite 3.53.4 (b09c88c) default core library. Untrusted SQL or a planted VIEW. load_extension stays off. Not RCE.	PUBLISHED 09 Oct 2026
UPDATED 09 Oct 2026	TAGS 0-day, sqlite, crash, collate, cwe-674, untrusted-sql

TL;DR

Lab files and proof-of-concept here.

01 Height 1, forever

I am @abraxas_null. The proof of concept is on GitHub: **abraxas/sqlite-collate-depth** (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** (Git `b09c88c14082339b66c7b7158d609a771e64ca69`, SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`). No CVE yet. CWE-674 / CWE-770. 7.5 High (`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H`). Local crash. Untrusted SQL. Default core library.

SQLite has a documented expression-depth limit. `SQLITE_MAX_EXPR_DEPTH` defaults to 1000. `sqlite3_limit(..., SQLITE_LIMIT_EXPR_DEPTH, ...)` can make it smaller. The comment in `sqliteLimit.h` says the point is hostile SQL. Most operators honor it. `COLLATE` does not.

The grammar action is three lines. It never calls `sqlite3ExprCheckHeight`. The new node keeps `nHeight = 1`. The walker still recurses on every `pLeft`. Fifty postfix `COLLATE BINARY` tokens return `1`. Eight thousand return `1`. Forty thousand return `1`, forty times past the documented cap. One hundred and twenty thousand ASan stack-overflow in `sqlite3WalkExprNN` and abort 134.

That is the leftover. Not a new opcode. Not a VDBE length integer. A postfix operator that forgot it was an expression tree.

02 Why SQLite after OpenLiteSpeed

I had just finished unpublished Crit/High hunts of Caddy, lighttpd, and OpenLiteSpeed. Web servers are a mood. Duplicate headers, TLS name matching, the 403 you thought you shipped. OpenLiteSpeed 1.9.3 still 403s a single `Cookie: litespeed_role=`. HTTP/1 duplicate Cookie skips that 403. That is a different post.

SQLite is the other end of the stack. It is not a daemon. It is a library that other daemons load, and it is the file format those daemons persist. sqlite.org/security.html is unusually honest for a C project of this age: untrusted SQL and untrusted `.db` (plus a sibling `-wal`) are first-class inputs. Memory corruption, crash, or heap disclosure on those inputs is a serious bug. Default `load_extension` is off. Typical amalgamation extras are FTS5, builtin JSON, rtree/geopoly. The CLI, `sqlite3_rsync`, zipfile, session, and official WASM are extra surfaces.

That is a better threat model than "only the C API is in scope." It is also a larger surface than people admit when they say SQLite is just a file.

I only wanted Critical or High. Local is in-scope. Medium stays on the floor unless it chains. This post is **G1 of fourteen leftovers** on the pin. The siblings are other notes. This one is the COLLATE chain.

03 Pinning 3.53.4, not the draft

Latest GA, not a draft. sqlite.org/download.html listed 3.53.4 (2026-07-24). 3.54.0 was a draft with a planned date of 2026-10-15. Drafts wait. Pinning a Fossil check-in that is not a release is how you "find" a bug the next tarball already closed.

Canonical tree is sqlite.org/src. GitHub sqlite/sqlite is a Fossil mirror. Official source zip is `sqlite-src-3530400`. Public domain. The Git tag is `version-3.53.4`. HEAD of that tag is `b09c88c14082339b66c7b7158d609a771e64ca69`. `SELECT sqlite_source_id()` on the amalgamation is `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`. The lab checks both before it feeds SQL.

3.53.4 itself is a maintenance wave over 3.53.0-3.53.3, a lot of it AI-discovered. That is not an insult. It is a skip list with extra steps. You do not pin 3.54.0 because the calendar looks newer.

Hwaci writes CVEs rarely. Developers fix in hours. The leftover class on this tree is "later trunk, not this GA." Forum bugs with a Fossil check-in after 2026-07-24 and no

CVE are the hunt, not the CVE table.

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. SQLite's table lives at sqlite.org/cves.html. On 3.53.4 the interesting ones were already closed:

- CVE-2026-11822 / CVE-2026-11824. FTS5 heap overflow when `SQLITE_DBCONFIG_DEFENSIVE` is off. `fts5LeafRead` `szLeaf<4`. Fixed 3.53.2. Duplicate CVEs for the same bug. In tree on this pin.
- CVE-2026-50812. Session extension NULL pointer deref in `sqlite3changeset_apply_v3` 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.
- CVE-2025-70873. zipfile CLI OOB on a malformed ZIP. Fixed 3.52.0. zipfile is a CLI extra, not the default library.
- CVE-2024-0232. JSON UAF from injected SQL. Fixed 3.43.2. Closed.
- Older FTS3 / printf / NULL rows, WAL-reset, C-API >2GB strings. Years before this pin.

Ada Logics `sqlite3_rsync` `readPow2` stack smash (`x>16`, replica 64KiB buffer vs origin page size) is in tree. Forum e2dfadf51209f12b. Trunk `805f3acd` 2026-07-09.

3.53.4 is 2026-07-24. Closed.

JFrog's "critical CVEs" that SQLite treated as unreproducible LLM slop stay on the floor. Third-party JDBC and LuxCal rows are other products.

What is *not* closed on this GA: later-trunk leftovers. WAL page-size mismatch uninitialized page tail. FTS5 secure-delete footer PutVarint. rtree `last_insert_rowid()==0` UAF. FULL JOIN USING authorizer skip. COLLATE height. CTE nest. VALUES UNION ALL error-path delete. Those are the fourteen. This post is the first.

05 Twenty hunts, one postfix operator

I split the tree. Parser / name resolution / expr / SELECT-DML / VDBE / IN / window. 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 SQLite said it would do.

Default `load_extension` is off. `TRUSTED_SCHEMA` is ON. Unauth remote RCE in libsqlite3 is not the product. ATTACH to any writable path is documented. `sqlite_log()` from a hostile schema is log injection, Medium, and DRH declined the backport. JSON CVE-2024-0232 class is closed. VDBE opcode-length integer overflow, Mem type-confusion, and IN-operator memory-safety leftovers did not meet the bar on this pin.

The parser hunt is where the temperature changed.

Ordinary binary operators go through `sqlite3PExpr()`. That allocator attaches the subtrees, sets `pRoot→nHeight` from the taller child plus one, and calls `sqlite3ExprCheckHeight()`. If the tree is deeper than `db->aLimit[SQLITE_LIMIT_EXPR_DEPTH]`, prepare fails with `Expression tree is too`

`large (maximum depth %d)`. Hostile `1+1+1+...` dies at 1000, or at 1 if you asked for 1.

`COLLATE` is not an ordinary binary operator in the grammar. It is a postfix:

```c
expr(A) ::= expr(A) COLLATE ids(C). {
  A = sqlite3ExprAddCollateToken(pParse, A, &C, 1);
}
```

Three lines in `src/parse.y`. No height. No check. The work is in `sqlite3ExprAddCollateToken`:

```c
Expr *sqlite3ExprAddCollateToken(
  const Parse *pParse,
  Expr *pExpr,
  const Token *pCollName,
  int dequote
){
  if( pCollName->n>0 ){
    Expr *pNew = sqlite3ExprAlloc(pParse->db, TK_COLLATE, pCollName, dequote);
    if( pNew ){
      pNew->pLeft = pExpr;
      pNew->flags |= EP_Collate|EP_Skip;
      pExpr = pNew;
    }
  }
  return pExpr;
}
```

`sqlite3ExprAlloc` sets `nHeight = 1` and never recomputes it. The prior expression becomes `pLeft`. Flags get `EP_Collate|EP_Skip`. `EP_Skip` is for affinity and `sqlite3ExprSkipCollate`. It is not a walker stop. `sqlite3WalkExprNN` looks at `EP_TokenOnly|EP_Leaf`, then recurses:

C Copy

```c
SQLITE_NOINLINE int sqlite3WalkExprNN(Walker *pWalker, Expr *pExpr){
  int rc;
  ...
    if( pExpr->pLeft && sqlite3WalkExprNN(pWalker, pExpr->pLeft) ){
      return WRC_Abort;
    }
```

N COLLATE nodes is N frames. `nHeight` stays 1 the whole way down, so `sqlite3ExprCheckHeight` is never even a candidate. `SQLITE_MAX_EXPR_DEPTH` compiled in at 1000 is a comment you can quote. It does not run.

`CAST` in the same file calls `sqlite3ExprAttachSubtrees`. COLLATE does not. That is the kind of inconsistency you hunt for after you have already read the skip list twice.

Later trunk noticed. Bug `2026-10-04T04:50:56Z`. Fixed trunk `252c0b7bf5` (2026-10-04). Not in this 3.53.4 tag. No CVE. Developers fix in hours. Releases wait.

06 What I almost filed instead

Pinning 3.54.0. The draft date is after the hunt. The leftover class is "GA still ships it." Drafts are not GA.

Treating `load_extension` as on. Default open is off. A finding that needs `sqlite3_enable_load_extension` is a different victim.

Calling this RCE. ASan died in `sqlite3WalkExprNN`. The process aborted. There is no demonstrated instruction-pointer control. Crash is the class. Inflating it is how you get ignored.

Treating `EP_Skip` as a walker skip. I read the flag name, I read `sqlite3ExprSkipCollate`, I almost closed the note. The walker does not consult `EP_Skip`. The chain is walked.

Filing the depth limit as "works." A 50-node chain returning `1` looks like a control. It is a control. It is also already under the documented cap. The leftover is the cap not running.

Re-filing CVE-2026-11824. `szLeaf<4` is in `fts5_index.c` on this pin. A different FTS5 leftover (secure-delete footer) is a sibling post, not this one.

Re-filing the rsync stack smash. `readPow2` is closed. The rsync SSH filename leftover is G12, another post.

Scoring COLLATE as parser DoS below High. sqlite.org/security.html says a crash on untrusted SQL is a serious bug. The documented defense is the depth limit. The operator skips the defense. That is High.

07 The lab

Host is macOS. One container.

Image built from official `sqlite-src-3530400.zip`, ASan CFLAGS (`-fsanitize=address -g -O1 -fno-omit-frame-pointer`), `configure --disable-tcl`, `make sqlite3`. Fallback clone of tag `version-3.53.4` if the zip 404s. Runtime is debian bookworm-slim plus `libasan8`. Compose name `sqlite-collate-depth`. Dummy `python3 -m http.server 18800` so `run.sh` has something to wait on. Bind `127.0.0.1:18800`. Work volume `./work:/work`. `ulimits.stack` 8 MiB, which is a normal C stack, not a trick.

The library is not a network daemon. The dummy port is a readiness gate. SQL is `docker compose exec sqlite3 -batch -noheader :memory: .read /work/....sql` with `ASAN_OPTIONS=detect_leaks=0:abort_on_error=1:halt_on_error=1`.

Pin check first: `SELECT sqlite_version()` must be `3.53.4`. `SELECT sqlite_source_id()` must match the Fossil id. Mismatch is FAIL. You do not crash a different amalgamation and call it this leftover.

Then the control: `SELECT 1` plus 50 `COLLATE BINARY`. 760 bytes. Returns `1`. If that fails, the grammar changed or ASan is already dying on a handshake. FAIL.

Then the crash attempts, in order: 8_000, 40_000, 120_000, 200_000. Each file is `SELECT 1` plus N copies of the token, written in 1000-token chunks so Python is not the bottleneck. Timeouts 90s under 80k, 180s above.

8_000 (120_010 bytes) returned `1`. Depth limit skipped. Stack survived.

40_000 (600_010 bytes) returned `1`. Same. Forty times the documented cap, still a row.

120_000 (1_800_010 bytes) died rc=134. ASan:

PLAIN TEXT Copy

```
AddressSanitizer:DEADLYSIGNAL
ERROR: AddressSanitizer: stack-overflow
    #0 ... in sqlite3WalkExprNN /src/sqlite/sqlite3.c:110283
SUMMARY: AddressSanitizer: stack-overflow /src/sqlite/sqlite3.c:110283 in sqlite3
ABORTING
```

That is SUCCESS. Not a 200k run. The first crash is the oracle.

Last line:

PLAIN TEXT Copy

```
SUCCESS sqlite-collate-depth how=asan-stack-overflow control=50 crash-n=120000 SC
```

Replay:

PLAIN TEXT Copy

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

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

08 Wrong turns, in the order I made them

Pinning the Fossil draft. 3.54.0 looks like "latest" on the timeline. It is not a release. GA is 3.53.4. The SOURCE_ID date is 2026-07-24.

Cloning GitHub as canonical. The useful tree is sqlite.org/src. The GitHub repo is a mirror. The lab prefers the official zip so the amalgamation is the one people actually compile.

Treating 8k as the crash. Eight thousand COLLATE tokens is already eight times `SQLITE_LIMIT_EXPR_DEPTH`. I expected "Expression tree is too large." I got `1`. That is the leftover asserting itself, and it is also not a crash. A lab that stops at "limit skipped" without a dead process is a note, not a Crash class.

Treating 40k as the crash. Same row. Same `1`. The 8 MiB stack still had room. ASan was quiet. I stared at `returned-row=1 (depth limit skipped, stack survived)` and considered FAIL. The next size exists because of that line.

Needing 120k. One hundred and twenty thousand tokens, 1.8 MB of SQL. Default `SQLITE_LIMIT_SQL_LENGTH` is 1e9, so length is not the defense. The C stack is. rc=134, `sqlite3WalkExprNN`, witness. If I had stopped at 40k I would have shipped a blog that said "the limit is skipped" and no ASan frame. That is a Medium-sounding sentence. The crash is the High.

A 200k first attempt. The PoC lists 200_000 as a last resort. 120k was enough. Do not wait for the last size if the third already aborted.

Expecting `EP_Skip` to save the walker. Flag names lie. `sqlite3ExprSkipCollate` honors it. `sqlite3WalkExprNN` does not.

Compiling without ASan. A non-ASan 120k chain can still SIGSEGV. The SUCCESS token is `how=asan-stack-overflow` with a frame in `sqlite3WalkExprNN`. The amalgamation in the Dockerfile is ASan on purpose.

A huge `ulimit -s`. Raising the stack until 120k survives proves you can hide a leftover with a knob. The lab uses 8 MiB.

A reverse shell. Theatre. The oracle is ASan aborting in the walker. Crash witness. No RIP payload.

Calling the dummy HTTP server the sink. Port 18800 is a compose readiness probe. The SQL is `:memory:` inside the container. Do not scan 18800 and call it a remote daemon.

Bundling G2 and G3. Nested CTEs skip `nNestSel`. VALUES UNION ALL skips the compound-SELECT cap and then dies in `sqlite3ExprDeleteNN`. Those are other leftovers, other compose projects, other loopback ports. This pack is COLLATE.

09 What an attacker can do

Feed untrusted SQL that is a long postfix `COLLATE BINARY` chain. Or plant that chain in a VIEW (or a trigger body) inside an untrusted `.db` and wait for the victim to `SELECT` it, run `PRAGMA integrity_check`, or otherwise walk the schema.

`SELECT 1 COLLATE BINARY COLLATE BINARY ...` is enough. Same shape:

PLAIN TEXT Copy

```
CREATE VIEW v AS SELECT 1 COLLATE BINARY COLLATE BINARY ...;
SELECT * FROM v;
```

The documented expression-depth limit never runs on this operator. Default SQL length easily holds the ~1.8 MB statement that overflowed an 8 MB C stack in this lab. The host process dies. Embedders that prepare attacker SQL in-process take the crash in their own address space. That includes the usual suspects: a web app that interpolates into sqlite3, a desktop app that opens a file from disk, a CLI pointed at a hostile database.

You do not get a shell. You do not get `load_extension`. You do not get a new opcode. You get a stack overflow in `sqlite3WalkExprNN` because a postfix collate node was born with height 1 and never grew up.

Reach is Local in the table because the library is not a listener. CVSS is AV:N because untrusted SQL is the sqlite.org/security.html input, and that SQL often arrived over a network the embedder already accepted. The lab is loopback. The class is Crash.

10 The fix

Call `sqlite3ExprCheckHeight` on the COLLATE node after attaching `pLeft`, the way `sqlite3PExpr` already does. Recompute `nHeight` from the child plus one. A postfix operator is still an expression tree.

Three lines today:

```c
expr(A) ::= expr(A) COLLATE ids(C). {
  A = sqlite3ExprAddCollateToken(pParse, A, &C, 1);
}
```

Those lines should leave a tree whose height is checked. `sqlite3ExprAddCollateToken` is the right function to do it. `EP_Skip` can stay. Affinity still wants to skip collate. The walker still has to be bounded.

Trunk `252c0b7bf5` (2026-10-04) is after this GA. 3.53.4 still ships the skip. Until a release with the height check, do not prepare untrusted SQL that you did not already cap some other way, and do not open a `.db` whose schema you did not build.

I am not going to print a 120_000-token one-liner you can paste at someone else's process. The missing `sqlite3ExprCheckHeight` is the useful part. If you own the box, run the proof of concept against loopback.

11 References

- Proof of concept: abraxas/sqlite-collate-depth · sqlite-collate-depth-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 · CWE-770
- CVSS **High 7.5** `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H`
- Tree: sqlite.org/src tag `version-3.53.4` (`b09c88c14082339b66c7b7158d609a771e64ca69`) · SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc` · official zip sqlite-src-3530400 · GitHub mirror sqlite/sqlite
- `src/parse.y` COLLATE action · `src/expr.c` `sqlite3ExprAddCollateToken`, `sqlite3ExprAlloc`, `sqlite3PExpr`, `sqlite3ExprCheckHeight` · `src/walker.c` `sqlite3WalkExprNN` · `src/sqliteLimit.h` `SQLITE_MAX_EXPR_DEPTH`
- sqlite.org/security.html · sqlite.org/cves.html
- Last lab line: `SUCCESS sqlite-collate-depth how=asan-stack-overflow control=50 crash-n=120000 SQLITE-COLLATE-DEPTH-WITNESS`



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