0-DAY NO CVE HIGH PUBLIC

SQLite VALUES UNION ALL cleanup crashes the host

SQLite 3.53.4 skips the compound-SELECT cap when SF_Values is set. WITH forces huge VALUES onto UNION ALL. EXPR_DEPTH rejects the tree, then recursive sqlite3ExprDeleteNN blows the C stack. Labbed 300000 rows, ASan stack-overflow. Not RCE. No CVE yet.

NAME SQLite VALUES UNION ALL cleanup crashes the host	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 VALUES compound cap. Default core library. Untrusted SQL.	PUBLISHED 09 Oct 2026
UPDATED 09 Oct 2026	TAGS 0-day, sqlite, crash, cwe-674, values, union-all

TL;DR

Lab files and proof-of-concept **here**.

01 The error path is supposed to be the safe path

I am @abraxas_null. The proof of concept is on GitHub: **abraxas/sqlite-values-union** (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`). 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`). Local. Crash. Default core library. Untrusted SQL.

Default `VALUES` is a co-routine. That path is cheap, iterative, and does not build a compound SELECT the size of the row list. Prefix the same statement with `WITH` and `sqlite3MultiValues` falls back to `UNION ALL` . Compound-SELECT length is skipped when `SF_Values` is set. `SQLITE_LIMIT_EXPR_DEPTH` (default 1000) then rejects the tree with `Expression tree is too large` . Cleanup of that tree recurses in `sqlite3ExprDeleteNN` until the C stack is gone. The documented limit fired. The destructor did not.

ASan on 3.53.4: 5000 plain rows return `5000` . `WITH` plus 1000 rows errors cleanly. `WITH` plus 300000 rows is stack-overflow in `sqlite3ExprDeleteNN` at amalgamation line 114219, process rc=134. Not RCE. SQLite does not eval the VALUES bytes.

02 Why SQLite

I had just finished unpublished Crit/High hunts of products that look like they should still be interesting and then spend a week proving the last High already shipped. Caddy v2.11.7 came back empty at this bar. lighttpd 1.4.85 on Unix was the same

kind of empty. OpenLiteSpeed still had a duplicate-Cookie leftover of 1.8.2.1. SQLite is the other end of that spectrum: a C library whose security model is honest, whose skip list is long, and whose latest GA still had later-trunk leftovers sitting in `parse.y`.

sqlite.org/security.html treats untrusted SQL and untrusted `.db` (plus a sibling `-wal`) as first-class inputs. Memory corruption, crash, or heap disclosure on those inputs is High. `load_extension` is off on default open. I only wanted Critical or High. Local is in-scope. Medium stays on the floor unless it chains. A finding on this hunt needed a leftover of a named 3.53 / 3.54 guard, a default-library crash on attacker SQL, or an authorizer skip the docs still sell as a column-level boundary.

Canonical tree is sqlite.org/src. GitHub is a Fossil mirror. Public domain. Developers fix in hours and do not write CVEs. The bugs forum is unpaid. I am not opening a public issue on sqlite/sqlite.

03 Pinning 3.53.4

Latest GA, not an RC. 3.54.0 was still a draft (planned 2026-10-15). Pinning a draft is how you "find" bugs that already have a check-in and no release. I did not.

GitHub tag `version-3.53.4` is commit `b09c88c14082339b66c7b7158d609a771e64ca69` (2026-07-24). Fossil `SQLITE_SOURCE_ID` is `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`. SHA3-256 of `sqlite3.c` is in the skip list. 3.53.4 itself is a maintenance patch for (mostly) AI-discovered bugs in 3.53.0-3.53.3. If a leftover of later trunk still fires here, it is not a stale minor.

The lab builds the canonical zip `https://www.sqlite.org/2026/sqlite-src-3530400.zip` with `-fsanitize=address`, not a distro package and not a GitHub tarball of a mirror.

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 `SQLITE_DBCONFIG_DEFENSIVE` is off. Fixed 3.53.2. Duplicate CVEs for the same bug. `fts5LeafRead` `szLeaf<4` is in this tree.
- CVE-2026-50812 session `sqlite3changeset_apply_v3` NPD on a malformed changeset. Medium. 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-2025-7458 integer overflow read. Fixed 3.42.0.
- CVE-2025-7709 FTS5 corrupt-index integer overflow. Fixed 3.50.3.
- CVE-2025-29088 / CVE-2025-52099 `sqlite3_db_config` LOOKASIDE OOB. Fixed 3.49.1.
- CVE-2025-70873 zipfile CLI OOB. zipfile is CLI extra, not default lib.
- CVE-2024-0232 JSON UAF. Fixed 3.43.2. Closed on this pin.
- CVE-2023-7104 session on a corrupted changeset. Needs `SQLITE_ENABLE_SESSION`.
- CVE-2022-35737 C-API array overflow on >2GB strings. Not reachable from SQL or a DB file.

- WAL-reset corruption (Tailscale race). Fixed 3.51.3 / 3.53.0. Closed on this pin.
- Ada Logics `sqlite3_rsync` stack smash (`readPow2` `x>16`). Trunk `805f3acd` 2026-07-09. This pin is 2026-07-24. In tree.

The 2026-51296 / 51297 / 51300 / 51302 / 51303 / 51304 family is unreproducible. SQLite and JFrog treat it as AI hallucination. I am not re-filing a ghost.

Third-party names that are not this library: SQLite JDBC, LuxCal, Node bindings, `.expert` CLI, `appendvfs`. Out.

Public forum leftovers of later trunk versus this GA were the hunt, not the skip: WAL page-size short-read, FTS5 secure-delete overrun, rtree UAF after `last_insert_rowid` 0, FULL JOIN USING authorizer, CLI `go` terminator, `.recover` schema, `sqlite_log` trusted_schema. Those are other posts or other surfaces. This post is the VALUES UNION ALL error path.

05 Twenty surfaces, three default crashes

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 3.53.4 said it would do.

Tokenizer token length is `i64`. `mxSqlLen` subtracts per token. Parser stack is `SQLITE_LIMIT_PARSER_DEPTH` (default 2500). No integer overflow leftover.

VDBE `growOpArray` sizes with `sqlite3_int64` and caps at `SQLITE_LIMIT_VDBE_OP`. `OP_Concat` sums in `i64` then checks `SQLITE_LIMIT_LENGTH`. Register VM, not a bytecode stack. No Mem type-confusion leftover reachable from untrusted SQL on this pin.

IN membership / eph-index / NOOP compare paths match the documented algorithm. Skip-list "LEFT JOIN IN" is a correctness class here, not memory-safety. Window functions: no integer-overflow / stack leftover at bar.

JSON CVE-2024-0232 class is closed. `load_extension` stays off. `sqlite_log()` from a hostile schema is log injection only (Medium; DRH declined backport). Generated-column `SQLITE_FUNCTION` authorizer is documented current behavior. `sqlite3MatchEName` nested-join crash wants `SQLITE_ALLOW_ROWID_IN_VIEW`, which is a non-default build. zipfile / kvvfs / CLI / WASM extras are not default lib.

Default library unpublished Crit/High is not NONE. Three stack-crash paths on untrusted SQL survived, plus a WAL info leak, two authorizer leftovers, FTS/rtree extras, session, and CLI/rsync. This post is G3.

G1 is COLLATE depth. Grammar action for `COLLATE` calls `sqlite3ExprAddCollateToken`, which never recomputes `nHeight` and never calls `sqlite3ExprCheckHeight()`. `SQLITE_LIMIT_EXPR_DEPTH` is the documented defense and does not run. A long postfix `COLLATE BINARY` chain dies in `sqlite3WalkExprNN`. Lab: 50 COLLATE returns 1; 120000 dies ASan stack-overflow. Not this post.

G2 is CTE nest. Nested VIEWs were capped on this pin (`nNestSel` versus `SQLITE_LIMIT_EXPR_DEPTH`). CTE attach at `select.c:5754` was not. Nested FROM at N=25000 is `Recursion limit` with no crash. Nested CTE at N=25000 is ASan stack-overflow in `selectExpander`. Not this post.

G3 is this VALUES leftover. The co-routine is the fast path. The UNION ALL fallback is the one that still builds a left-recursive compound the size of the row list, skips the compound cap because the flags say VALUES, then deletes the tree with a recursive destructor after EXPR_DEPTH has already said no.

06 Two compilers, one skip

`sqlite3MultiValues` has two ways to represent a multi-row `VALUES` list. The comment in `insert.c` is clearer than most advisories.

Co-routine: each row is coded into a VM as it is parsed. The returned Select is `SELECT * FROM (co-routine)`. Iterative. Cheap. This is the default.

UNION ALL fallback: the returned Select is `pLeft UNION ALL SELECT pRow`. A 300000-row VALUES clause is a 300000-term compound. The comment even says it "may consume a lot of memory."

The fallback is not a bug by itself. The co-routine cannot be used when:

- the statement has a `WITH` (two cursors cannot iterate the same hard-coded Yield register)
- the schema is being parsed (`db→init.busy` : VIEW / TRIGGER, no VM yet)
- a row is non-constant
- the first row has affinity (`CAST`)
- `IN_SPECIAL_PARSE`

`WITH q AS (SELECT 1) SELECT count(*) FROM (VALUES (0),(0),...)` is enough. You do not need a hostile VIEW. You do not need CAST. You need the prefix that turns the co-routine off.

Then the compound-SELECT cap, default 500 (`SQLITE_MAX_COMPOUND_SELECT`), is supposed to refuse a 501-term UNION. For VALUES it does not:

```c
if( (p→selFlags & (SF_MultiValue|SF_Values))==0
 && (mxSelect = pParse→db→aLimit[SQLITE_LIMIT_COMPOUND_SELECT])>0
 && cnt>mxSelect
){
  sqlite3ErrorMsg(pParse, "too many terms in compound SELECT");
}
```

`SF_Values` is set on the fallback Select. The `==0` test is the skip. 300000 terms is legal as far as this check is concerned.

EXPR_DEPTH still runs. Default 1000. At 1000 rows the parse errors `Expression tree is too large (maximum depth 1000)` and the process lives. That is the control that looks like a fix until you keep going.

`clearSelect()` walks `pPrior` in a `while` loop. Iterative. Fine. Then it calls `sqlite3ExprListDelete` on each term's expression list. `sqlite3ExprDeleteNN` still recurses on `pRight`, and on `pLeft` when `EP_Static` is set:

C Copy

```c
if( p->pRight ){
  sqlite3ExprDeleteNN(db, p->pRight);
}
...
if( p->pLeft && p->op!=TK_SELECT_COLUMN ){
  Expr *pLeft = p->pLeft;
  if( !ExprHasProperty(p, EP_Static)
   && !ExprHasProperty(pLeft, EP_Static)
  ){
    sqlite3DbNNFreeNN(db, p);
    p = pLeft;
    goto exprDeleteRestart;
  }else{
    sqlite3ExprDeleteNN(db, pLeft);
  }
}
```

Unary operators get a `goto` so they do not eat a frame. The VALUES UNION ALL tree is not that shape. The error-cleanup path is a recursive walk of a tree the parser just refused for being too deep. C stack on the lab image is 8 MiB. 300000 rows is enough.

Public report on later trunk is Bug `2026-10-04T05:34:47Z` (default limits, stock shell). No check-in on this pin. No CVE.

07 What I almost filed instead

Collate and CTE as this bug. They are sibling default crashes on the same hunt. Different sinks. COLLATE never calls `sqlite3ExprCheckHeight`. CTE attach never increments `nNestSel`. This leftover is the VALUES compound skip plus a recursive destructor on the error path. Three findings, three labs, three posts.

FULL JOIN USING authorizer. Skip-list leftover, still present, Privilege leftover, needs an embedder `sqlite3_set_authorizer`. Not a crash. Not this class.

xferOptimization authorizer skip. DRH treated `d1cdb817` as a 3.54 feature, not a 3.53 bugfix. "Not going onto branch-3.53." Sibling of the JOIN leftover. Below bar as a vendor-declared feature.

`sqlite3SchemaToIndex` **unbounded loop.** Public Bug `2026-10-03T19:21:58Z`, heap-buffer-overflow after `sqlite3_deserialize` plus an eponymous vtab. C-API deserialize, not this parser path.

Pinning 3.54.0. Draft. The leftover of later trunk is the point of pinning GA.

Calling it RCE. ASan stack-overflow on a destructor is a crash. There is no RIP payload in this pack. There is no "then we exec." The host dies. That is the finding.

Filing CVE-2026-11824 again. FTS5 `szLeaf<4` is in 3.53.4. Different extension, different year, different advisory.

Hallucinated 2026-51296. If your scanner emitted a Critical with no repro, it is not this.

08 The lab

Host is macOS. One container. Compose project `sqlite-values-union`. Loopback `127.0.0.1:18820`. Image is debian bookworm, canonical 3.53.4 source zip, `CFLAGS="-g -O1 -fno-omit-frame-pointer -fsanitize=address"`. `ulimits.stack`

8 MiB. `mem_limit` 4g. `ASAN_OPTIONS=detect_leaks=0:abort_on_error=1:halt_on_error=1`. The published port is a dummy `python3 -m http.server` so the PoC can wait-ready without talking SQL over TCP. SQL goes through `docker compose exec sqlite3 -bail :memory:`.

The PoC pins `sqlite_version()` and `sqlite_source_id()` against 3.53.4 and the Fossil SOURCE_ID above. Then four statements:

01 Plain `SELECT count(*) FROM (VALUES (0), ...)` at 5000 rows, no WITH. Must return 5000. Co-routine path. If this crashes, the lab is wrong.

02 `WITH q AS (SELECT 1)` plus 100 rows. Must return 100. Fallback path, small.

03 Same WITH at 1000 rows. Must error `Expression tree is too large` and must not crash. That is the documented cap doing its job on the fallback path.

04 Same WITH at 300000 rows (then 320000, then 400000 if 300000 somehow lives). Must ASan stack-overflow.

Last run:

PLAIN TEXT Copy

```
pin version=3.53.4 source_id=2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab397887
sql-result rows=5000 with_cte=0 rc=0 crashed=0 how=exit-0 stdout='5000'
sql-result rows=100 with_cte=1 rc=0 crashed=0 how=exit-0 stdout='100'
sql-result rows=1000 with_cte=1 rc=1 crashed=0 how=exit-1
  stderr='Parse error near line 1: Expression tree is too large (maximum depth 10
sql-result rows=300000 with_cte=1 rc=134 crashed=1 how=asan-stack-overflow
  sqlite3ExprDeleteNN /src/sqlite/sqlite3.c:114219
SUCCESS sqlite-values-union how=with-values-union-all-fallback-exprdelete rows=30
```

Replay:

```
PLAIN TEXT
Copy

1 git clone https://github.com/abraxas/sqlite-values-union
2 cd sqlite-values-union/lab
3 ./run.sh
```

Bind it to loopback. The loopback client is `lab/run.sh` with a banner on it. Do not `docker compose up` this stack next to another SQLite leftover on the same port.

09 Wrong turns, in the order I made them

1000 rows. EXPR_DEPTH default is 1000. WITH plus 1000 VALUES rows is the textbook reject. `Parse error near line 1: Expression tree is too large (maximum depth 1000)`. rc=1. No ASan. No deadlysignal. I almost wrote FAIL and moved on. The leftover is not "the depth cap is missing." The leftover is "the depth cap fires, then cleanup of the tree it just refused walks off the C stack." You need enough rows that destructor depth exceeds 8 MiB. 300000 did. 1000 did not.

Plain VALUES as the crash. 5000 rows without WITH stays on the co-routine, returns 5000, lives. If you omit `WITH`, you are not on this path. The comment in `insert.c` is the whole trigger: `pParse→bHasWith`.

Schema-only. `db→init.busy` takes the same fallback, so a VIEW or TRIGGER body with a huge VALUES list is in-scope. I did not need it. WITH on a one-shot `:memory:` connection is enough and is the smaller lab.

CAST on row 1. That is fallback condition (d). It works. It is also a more clever SQL than the hunt needed. WITH is the boring prefix.

Treating compound-SELECT 500 as the oracle. Default `SQLITE_MAX_COMPOUND_SELECT` is 500. A 501-term `SELECT 1 UNION ALL SELECT 1 UNION ALL ...` is `too many terms in compound SELECT`. VALUES is excluded from

that check on purpose (the flags). Using a normal UNION as the crash witness tests the cap that still works.

Amalgamation line numbers as source lines. ASan prints `sqlite3.c:114219`. The pin source is `expr.c` `sqlite3ExprDeleteNN`. The pack names `parse.y:562-567` and `insert.c:681-704`. Do not file the amalgamation as the sink.

Wine, a network listener, a second container. This is local untrusted SQL. The dummy HTTP server is only wait-ready. SQL is `compose exec`. Adding a TCP SQL protocol would have tested a protocol that SQLite does not ship.

OOM as SUCCESS. rc=137 is the 4g `mem_limit`, not a stack crash. The PoC skips that candidate and tries a larger row count, then FAILs if nothing ASan-overflows. 300000 was overflow, not OOM.

Calling it a parser-stack leftover. Parser depth is 2500 and is not this. Compound SELECT recursion in the code generator is the reason the 500 cap exists, and VALUES skips that cap, but the death in the lab is the Expr destructor after EXPR_DEPTH, not `sqlite3SelectNew` recursion.

10 What an attacker can do

Get SQL into `sqlite3_prepare` / the shell / an app that prepares attacker text. Prefix a huge `VALUES` list with `WITH` (or plant the same list in a VIEW / TRIGGER so schema reload takes the fallback). The prepare is supposed to fail with an expression-depth error. Cleanup of that failure can abort the host.

The victim is the process that embedded SQLite: a CLI, a desktop app, a service that opens a hostile `.db`, a multi-tenant host that prepares tenant SQL. Reach is local in the library sense. CVSS still uses AV:N because that is how a crash in a library that network apps embed is scored when the SQL arrives from somewhere the app already accepts. Integrity is none. Confidentiality is none. Availability is the process.

This is not remote unauthenticated RCE against `libsqlite3`. There is no `load_extension` in the lab. There is no shell. There is no RIP control. ASan stack-overflow on `sqlite3ExprDeleteNN` is the witness. If your threat model is "untrusted SQL must not kill the host," this leftover is in it. If your threat model is "untrusted SQL is already game over," you are not the audience for sqlite.org/security.html 1.1, and you can stop here.

11 The fix

Apply `SQLITE_LIMIT_COMPOUND_SELECT` to VALUES the same way other compounds are capped, or stop building a left-recursive UNION ALL the size of the row list when the co-routine is unavailable. Walk the error-cleanup path iteratively. Unary-style `goto exprDeleteRestart` already exists for `pLeft` without `EP_Static`. The VALUES fallback tree does not take it.

Later trunk (Bug `2026-10-04T05:34:47Z`) is after this GA. 3.54.0 was still draft when I pinned. I am not going to pretend a draft check-in is a release.

Until then, do not prepare untrusted SQL that can carry an arbitrary-size `VALUES` list on a connection that also has `WITH`, and do not load schema from people you do not trust. That sentence is also the product's security model. The leftover is that the model already has a depth cap, and the destructor of the capped tree can still take the process with it.

I am not going to print a 1.2 MB VALUES statement you can paste at someone else's CLI. The skip in `parse.y` is the useful part. If you own the host, run the proof of concept against loopback.

12 References

- Proof of concept: abraxas/sqlite-values-union · sqlite-values-union-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
- Tree: sqlite.org/src · github.com/sqlite/sqlite version-3.53.4 (`b09c88c14082339b66c7b7158d609a771e64ca69`) · SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`
- `src/parse.y:562-567` compound-SELECT skip when `SF_Values` is set · `src/insert.c:681-704` `sqlite3MultiValues` UNION ALL fallback · `src/expr.c:1406-1430` recursive `sqlite3ExprDeleteNN` · `src/select.c:82-107` iterative `clearSelect`
- sqlite.org/cves.html · sqlite.org/security.html · sqlite.org/forum

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