0-DAY NO CVE HIGH PUBLIC

SQLite 3.53.4 CLI, .recover emits hostile schema as a later dot-command

SQLite 3.53.4 sqlite3 .recover still prints a planted sqlite_schema.sql CREATE plus a later .shell. recoverOneStmt prepares only the first statement. The documented restore pipe runs the dot-command as the operator UID. Labbed on 3.53.4. No CVE yet.

NAME SQLite 3.53.4 CLI, .recover emits hostile schema as a later dot-command	TYPE 0-day analysis
CVE n/a	CVE RISK high
DISCLOSURE STATUS public	VENDOR SQLite (Hwaci)
AFFECTED SQLite 3.53.4 (b09c88c) sqlite3 CLI .recover / .dump. SOURCE_ID 2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc. Documented recover pipe. Not libsqlite3. Not --safe.	PUBLISHED 09 Oct 2026
UPDATED 09 Oct 2026	TAGS 0-day, sqlite, command-injection, recover, cwe-78, local

TL;DR

Lab files and proof-of-concept here.

01 The restore pipe is the victim

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

This is **SQLite 3.53.4** (`b09c88c14082339b66c7b7158d609a771e64ca69`). Fossil SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc` . No CVE yet. CWE-78. 7.8 High (`CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H`). Local. CLI-gated. The victim action is the documented recover pipe, not a pasted `go` line.

sqlite.org/recovery.html tells you how to rescue a corrupt database with the CLI:

PLAIN TEXT Copy

```
sqlite3 corrupt.db .recover >data.sql
sqlite3 recovered.db <data.sql
```

That is two sqlite3 processes. The first one prints SQL. The second one reads it. Input lines that begin with a dot are not SQL. They are intercepted by sqlite3 and become meta-commands. `.shell` is `system()` .

`.recover` is supposed to reconstruct CREATE statements from `sqlite_schema.sql` . `recoverOneStmt` prepares only the first statement on the recovery handle, so a planted newline plus `.shell` is not executed there. It is still printed. The restore sqlite3 is a normal CLI. It sees a line that starts with `.` and runs it as the operator UID.

The lab oracle is `echo` of `SQLITE-CLI-RECOVER-WITNESS`. Not a reverse shell.

02 Why SQLite

I had just finished unpublished Crit/High hunts of HTTP servers and a pile of application stacks. SQLite is the other kind of tree: public domain, latest GA, untrusted SQL and untrusted `.db` are first-class inputs per sqlite.org/security.html, and the developers fix real bugs in hours and do not write CVEs. That is a good hunt. It is also a hunt where you can waste a week re-filing last month's forum thread.

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, crash, or heap disclosure on untrusted SQL or an untrusted database, an authorizer skip, or a CLI extra that turns a hostile `.db` into a host command. `load_extension` off is the default and is not a leftover.

The library and the CLI are not the same product. People embed libsqlite3. People also run `sqlite3` on files they did not write. `.recover` exists because those files sometimes come back damaged. The docs tell you to pipe the output into a second sqlite3. That pipe is in scope the moment the first process will print a dot-command the second process will honor.

03 Pinning 3.53.4

Latest GA. Not an RC. Not 3.54.0. 3.54.0 was still a draft with a planned 2026-10-15 date. Pinning a draft is how you "discover" a bug that trunk already closed, or miss a leftover that only still lives on the release people actually ship.

GitHub's `sqlite/sqlite` is a Fossil mirror. Canonical is sqlite.org/src. The tag is **version-3.53.4**. HEAD landed on `b09c88c14082339b66c7b7158d609a771e64ca69`. `sqlite3 --version` prints the SOURCE_ID above. That is the pin.

Unix CLI recover is not a leftover compile flag you have to hunt for. `OPT_SHELL` includes `SQLITE_ENABLE_DBPAGE_VTAB`. Default `./configure && make sqlite3` has `.help recover`. The lab builds sqlite-src-3530400 that way.

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 `szLeaf<4` heap overflow. Fixed 3.53.2. `fts5LeafRead` has the check.
- CVE-2026-50812 session NPD on a malformed changeset. Optional `-DSQLITE_ENABLE_SESSION`.
- CVE-2025-70873 zipfile CLI OOB. Fixed 3.52.0. Remaining zipfile UAF needs attacker UPDATE SQL, not `.recover`.
- CVE-2022-46908 CLI `--safe`. `.shell`, `.read`, `.import`, `.archive`, `.output` still `failIfSafeMode`. Default sqlite3 does not turn `--safe` on. The documented recover pipe does not pass it.
- CVE-2022-35737 C-API >2GB strings. Not reachable from SQL or a DB file.
- CVE-2024-0232 JSON UAF. Closed. concat_ws / LOOKASIDE / FTS5-index integer overflows from 3.49-3.50 are closed.
- Ada Logics `sqlite3_rsync` stack smash (`readPow2` `x>16`) is in this pin.
- sqlite.org/cves.html "AI hallucination" CVE block (51296 and friends): skip.

- `.expert` CVE-2021-36690 and appendvfs CVE-2021-31239: skip.

The 3.53.4 AI-bug maintenance wave is in tree. I am not re-filing it.

05 Twenty surfaces, then the CLI extra

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.

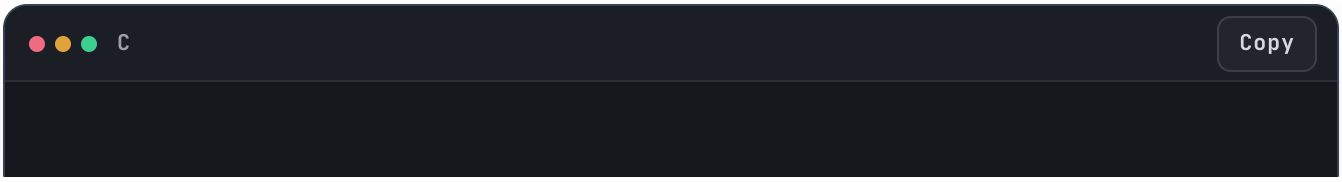
Default library `load_extension` stays off. TRUSTED_SCHEMA is ON; DIRECTONLY still blocks the dangerous functions from a hostile VIEW. FTS5 `szLeaf<4` holds. JSON subtype walk holds. Authorizer is off unless the embedder installs one. Official WASM is an extra. Session is a compile extra.

The default library still has High leftovers on this pin that are not this post: COLLATE depth crash, CTE nest crash, VALUES UNION ALL crash, WAL page-size mismatch uninit page tail, FULL JOIN USING and xferOptimization authorizer skips, FTS5 / FTS3 / rtree crash and UAF extras, session nCol heap write, sqlite3_rsync SSH filename, CLI `go` / `/` terminator. Those stand alone. This post is G14.

Recover's own hunt almost buried this one.

06 recoverOneStmt looks like a strip. It is a print leak.

`ext/recover/sqlite3recover.c` is honest about the first statement:



```c
static int recoverOneStmt(sqlite3 *db, const char *zSql){
  sqlite3_stmt *pStmt = 0;
  int rc;
  if( zSql==0 ) return SQLITE_OK;
  rc = sqlite3_prepare_v2(db, zSql, -1, &pStmt, 0);
  /* steps and finalizes the first statement only */
}
```

The comment says if `zSql` contains two or more SQL statements separated by `;`, only the first is run. A hostile `sqlite_schema.sql` of `CREATE TABLE t(x);\n.shell echo WITNESS` prepares `CREATE TABLE t(x)`. The `.shell` line is not SQL, so it is not a second statement on that handle. Recovery continues. The extra text is still in `zSql`.

Then the write path:

C Copy

```c
rc = recoverOneStmt(p->dbOut, zSql);
if( rc==SQLITE_OK ){
  recoverSqlCallback(p, zSql);
```

And the CLI callback:

C Copy

```c
static int recoverSqlCb(void *pCtx, const char *zSql){
  ShellState *pState = (ShellState*)pCtx;
  cli_printf(pState->out, "%s;\n", zSql);
  return SQLITE_OK;
}
```

`%s` plus a semicolon. The full column, including the newline and the dot-command. `recoverDatabaseCmd` also prints `.dbconfig defensive off` first, because the restore has to be allowed to write schema. That is the documented shape of the

script. It is also the shape that makes a later `.shell` legal input for the second sqlite3.

`.dump` is the same emit class for ordinary CREATE. `printSchemaLine` is `%s` plus `;\n`. Virtual-table dump uses `%q` (quoted). Ordinary table SQL is raw schema text. Plant the same CREATE-plus-newline-`.shell` and `.dump` will print it too. The lab used `.recover` because that is the documented rescue pipe.

`--lost-and-found` interpolates the CLI argument unquoted into `CREATE TABLE %s(%s)`. That is operator-controlled. Below this finding.

07 How you plant the schema

Default CLI defensive mode makes `writable_schema` a no-op. The plant is not "run SQL against a locked-down CLI." It is "give the victim a `.db` whose `sqlite_schema.sql` column already has the extra line." Building that file in the lab takes:

PLAIN TEXT Copy

```
.dbconfig defensive off
.dbconfig writable_schema on
CREATE TABLE t(x);
PRAGMA writable_schema=ON;
UPDATE sqlite_schema SET sql = 'CREATE TABLE t(x);
.shell echo SQLITE-CLI-RECOVER-WITNESS' WHERE name='t';
```

A clean control plant of `CREATE TABLE t(x);` recovers without `.shell`. The hostile plant recovers with `.shell` still in the script. That is the split.

The attacker does not need the victim to type those pragmas. The attacker needs the victim to recover the file.

08 Lab

Compose `sqlite-cli-recover`. Loopback `127.0.0.1:18930`. Debian bookworm, sqlite-src-3530400, default Unix CLI. Health listener on 18930 so `run.sh` can wait without racing the build. `lab_health.py` is in the pack because the image copies it.

Control `.recover` of a clean schema: no `.shell` line.

Hostile plant: `CREATE TABLE t(x);` plus newline `.shell echo SQLITE-CLI-RECOVER-WITNESS`. `.recover` printed:

PLAIN TEXT Copy

```
.dbconfig defensive off
BEGIN;
PRAGMA writable_schema = on;
...
CREATE TABLE t(x);
.shell echo SQLITE-CLI-RECOVER-WITNESS;
PRAGMA writable_schema = off;
COMMIT;
```

Piped into a second sqlite3 on `restored.db`. stdout:

PLAIN TEXT Copy

```
          defensive off
SQLITE-CLI-RECOVER-WITNESS
```

Last line:

PLAIN TEXT Copy

```
SUCCESS sqlite-cli-recover how=recover-pipe-dot-shell recover-has-shell=yes restc
```

Replay:

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

Or from a fresh clone:

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

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

`.shell` is `system()` after `failIfSafeMode`. The restore sqlite3 in the documented pipe is not `--safe`. The oracle is echo. I am not going to print a reverse shell and pretend that made the CVSS more honest.

09 Wrong turns, in the order I made them

Pinning 3.54.0. Draft. Planned mid-October. The leftover that still fires on latest GA is the one operators have. 3.54 extras that do not exist on this pin (`.diskused`, NewVuln4 contentless totals) are not this bug.

Calling recover's first-statement prepare a strip. The recover/misc hunt scored the extra text as not executed on `dbOut` and walked away with NONE. That sentence is true and incomplete. The CLI prints `zSql` after the prepare succeeds. The victim of `.recover` is not the recovery handle. It is the next sqlite3.

Re-filing CVE-2022-46908. `--safe` still blocks `.shell`. Default sqlite3 does not turn it on. The recovery docs do not mention it. A leftover that only fires when someone forgets a flag they were never told to set is a different paper. This leftover fires on the command sqlite.org prints.

Treating the Python `sqlite3` module as the sink. The module does not implement `.recover` or `.shell`. The sink is the CLI. A lab that imports the stdlib and never execs `sqlite3` is a lab of nothing.

Expecting hostile SQL in a VIEW to call `.shell`. Dot-commands are not SQL. They do not run inside VDBE. They run when the CLI sees a line that starts with `.` and the SQL buffer is empty. The plant has to survive as text in the restore script, not as an expression the library evaluates.

Expecting `writable_schema` alone to plant. Defensive is on by default in the CLI. `writable_schema` is a no-op until `.dbconfig defensive off`. The lab records that order so nobody "reproduces" a no-op and calls the finding FAIL.

Calling this libsqlite3 RCE. Default amalgamated library without the shell is not this bug. `OPT_SHELL` / `SQLITE_ENABLE_DBPAGE_VTAB` is typical Unix `sqlite3`. Embedders who never ship the CLI are not the victim model. Operators who run `sqlite3 hostile.db .recover | sqlite3 restored.db` are.

Scoring `.dump` as a different bug and then dropping `.recover`. Same emit class. `%s` plus semicolon. Virtual-table dump already quotes. Ordinary CREATE does not. One leftover, two commands. The documented victim action is recover.

Putting a reverse shell in the oracle. Theatre. `echo WITNESS` is enough to prove `system()`. Anything else is a payload, and this pack does not ship payloads.

Treating `--lost-and-found` as the High. Operator argv. If you already control the command line you already have `.shell`. The leftover is schema bytes from a file you did not write.

Filing zipfile UAF, QRF control characters, or sqlite3_analyzer `*/` comment injection as this class. Different trees, different bars, some already public. Analyzer is not sqlite3. QRF is display. zipfile UAF needs attacker UPDATE SQL.

10 What an attacker can do

Give the victim a SQLite database. Plant `sqlite_schema.sql` as a valid first `CREATE TABLE` plus a newline plus `.shell` (or `.system`, `.output`, `.read`). A victim who recovers that database with the documented pipe runs the dot-command as their UID.

No login. No network listener on sqlite3. Local, UI:R: the operator has to point sqlite3 at the file and restore. That is the same victim model as "open this backup." The backup tool is sqlite3. The restore format is a CLI script that still honors dot-commands.

`.shell CMD` is `system()`. Integrity, confidentiality, and availability of that UID are all in play. The lab proves command execution with echo. `.output` / `.once` are arbitrary file write. `.read` of a second script is more of the same. `--safe` on the restore process blocks those sinks. It is not in the documented command.

Not remote unauth RCE in libsqlite3. Not a crash. Not an info leak of page-cache tail. Command injection in the CLI restore path. CWE-78. High 7.8, AV:L, UI:R, PR:N.

I am not going to print a pipeline you can paste at someone else's home directory. The missing strip after `recoverOneStmt` is the useful part. If you own the files, run the proof of concept against loopback.

11 The fix

Do not print schema SQL that is not a single statement. After `sqlite3_prepare_v2`, emit only the bytes of the prepared statement, or refuse the row if `pzTail` is not empty and not whitespace. Drop a later line that starts with `.`. Quote ordinary CREATE the way virtual-table dump already uses `%q`, so a newline cannot become a new CLI line.

Until then, recover into a file, read the file, and do not pipe it into sqlite3 unless the restore process is `--safe`. `--safe` is a mitigation, not a reason to leave `%s` in `recoverSqlCb`. The docs should not recommend a pipe that runs host commands from schema text.

`.dump` needs the same strip. Same emit, same plant.

12 References

- Proof of concept: abraxas/sqlite-cli-recover · sqlite-cli-recover-Abraxas-Labs.py
- Lab: `lab/` · Dockerfile · docker-compose.yml · run.sh · poc.py · lab_health.py
- @abraxas_null · github.com/abraxas · abraxaslabs.tech · abraxas.null@proton.me
- CWE-78 · 7.8 High
- Tree: sqlite.org/src tag version-3.53.4 (`b09c88c14082339b66c7b7158d609a771e64ca69`) · SOURCE_ID `2026-07-24 19:02:57 bf7c7f30031888f4e796e429ab3978879485813aaca6f641c7b33e4e09459bcc`
- `recoverOneStmt` · `recoverWriteSchema1` · `recoverSqlCb` · `printSchemaLine` · `.shell` `system()`
- Recovering Data From A Corrupt SQLite Database · Command Line Shell For SQLite · 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**

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