0-DAY NO CVE HIGH PUBLIC

Uptime Kuma, unauthenticated setup TOCTOU

Uptime Kuma 2.5.5 Socket.IO setup on first-run COUNTs users, bcrypts, then INSERTs. Username is UNIQUE only. Concurrent unauth clients can insert a hidden second admin, then disableAuth auto-logins as user id 1. No CVE yet.

NAME Uptime Kuma, unauthenticated setup TOCTOU	TYPE 0-day analysis
CVE n/a	CVE RISK high
DISCLOSURE STATUS public	VENDOR Uptime Kuma
AFFECTED Uptime Kuma through 2.5.5 (c98982a); unpublished. First-run empty user table only. Lab image louislam/uptime-kuma:2.5.5. Sequential late setup is rejected.	PUBLISHED 29 Sept 2026
UPDATED 29 Sept 2026	TAGS 0-day, uptime-kuma, toctou, ato, cwe-362, cwe-367, unauthenticated

TL;DR

Lab files and proof-of-concept here.

01 COUNT is not a lock

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

This is **Uptime Kuma 2.5.5** (`c98982a`), Louis Lam. No CVE yet. CWE-362 / CWE-367. 7.4 High (`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N`).

02 How I found it

I started with SECURITY.md and the published GitHub advisories, plus the public Matomo `siteId` XSS and the wontfix cloud-metadata SSRF. Then I went through tag **2.5.5** (`c98982a`): auth/setup/2FA, socket IDOR, HTTP/monitor SSRF, SSTI leftovers, path/LFI, real-browser, plugin/tailscale, status-page, docker.sock, prototype pollution, push/prometheus, database, monitor exec, analytics XSS, settings, websocket CSRF, frontend XSS, jobs, Apprise, maintenance, master-delta. Most of that returned nothing new. Auth did.

First-user setup is a public Socket.IO event. No login. `setup` does `COUNT(user)`, then `await passwordHash.generate`, then `R.store`. `passwordHash.generate` is `bcrypt.hash` cost **10**. That is a nap, not a lock. Two already-connected sockets that both see `count == 0` both hash, both insert. There is no transaction. No mutex. `needSetup` is in-memory and only used to emit `"setup"` on connect. The write path ignores it and re-counts.

The schema is username UNIQUE only. `knex_init_db.js` does not enforce one row. The product still assumes a single admin (multi-admin is unmerged). There is no user-list UI, so a second row is invisible. Same username collides (`SQLITE_CONSTRAINT`). Different usernames both land.

I stood up `louislam/uptime-kuma:2.5.5` on loopback with `UPTIME_KUMA_DB_TYPE=sqlite` so the v2 database wizard is skipped and first-run is this `setup` path. bcrypt cost 10 is a race, not a guarantee on the first try. `run.sh` wipes the volume and retries. Exit 2 is a miss. Fresh empty `user` table next.

When it hit: sqlite `1|labadmin` and `2|UK-SETUP-TOCTOU-WITNESS`. Sequential late setup: `Uptime Kuma has been initialized`. Extra user logged in, `setSettings` `disableAuth=true`. Reconnect auto-login is `R.findOne("user")`: lowest id, the operator. `prepare2FA` otpauth identity is `labadmin`. I am not reprinting the URI.

Nearby GHSA-23q2-5gf8-gjpp is disableAuth **re-enable** failing to drop sockets. Different bug.

I am not printing a Socket.IO `setup` burst you can paste at someone else's first-run dashboard. The COUNT-then-bcrypt-then-INSERT is the useful part.

Wrong turns already recorded: one user row after concurrent setup (race miss, not a patch); `UK-SETUP-TOCTOU-WITNESS` missing because both sockets used the **same** name; treating dashboard HTML 200 as SUCCESS; a reverse shell. Theatre. The oracle is two sqlite rows, then `disableAuth` auto-login as id 1.

03 Check, hash, insert

`setup` is public:

JAVASCRIPT Copy

```
socket.on("setup", async (username, password, callback) => {
    try {
        if (passwordStrength(password).value === "Too weak") {
            throw new TranslatableError("passwordTooWeak");
        }

        if ((await R.knex("user").count("id as count").first()).count !== 0) {
            throw new Error(
                "Uptime Kuma has been initialized. If you want to run setup agai
            );
```

```
        }

        let user = R.dispense("user");
        user.username = username;
        user.password = await passwordHash.generate(password);
        await R.store(user);

        needSetup = false;
```

The schema:

JAVASCRIPT Copy

```
await knex.schema.createTable("user", (table) => {
    table.increments("id");
    table.string("username", 255).notNullable().unique().collate("utf8_general_ci
    table.string("password", 255);
```

After the extra admin sets `disableAuth`:

JAVASCRIPT Copy

```
if (await setting("disableAuth")) {
    log.info("auth", "Disabled Auth: auto login to admin");
    await afterLogin(socket, await R.findOne("user"));
    socket.emit("autoLogin");
}
```

`findOne` is the lowest id. Hidden user 2 turns auth off. New sockets become user 1.

04 What an attacker can do

Hit a freshly started Kuma at the same time the operator submits `/setup`. Default Docker bind is `0.0.0.0:3001`. Insert a second `user` row with a different username. There is no user-list UI, so the extra account is invisible. Log in as that user, set

`disableAuth`, and ride `R.findOne("user")` auto-login onto the operator's monitors, notifications, API keys, and settings.

After the first row commits, a later sequential `setup` is rejected. This is not "setup stays open." It is a race during the bcrypt window. Window is first-run only. Already-initialized instances with an admin are not this bug.

A latent API-key enable/disable IDOR (`UPDATE api_key SET active=? WHERE id=?` with no `user_id`) also becomes live once two users exist. That is a follow-on of the extra row, not the lab oracle.

05 The lab (run this at home)

Source of truth is `lab/`. Image `louislam/uptime-kuma:2.5.5`. Port **18141**. Bind it to loopback. `UPTIME_KUMA_DB_TYPE=sqlite` skips the v2 database wizard so first-run is this `setup` path. Product default listen is `0.0.0.0:3001`. Do not publish that.

DOCKERFILE Copy

```
# Loopback lab image pin for uptime-kuma-setup-toctou. Full stack: docker-compose
FROM louislam/uptime-kuma:2.5.5
```

`docker-compose.yml`:

YAML Copy

```yaml
name: uptime-kuma-setup-toctou

services:
  kuma:
    image: louislam/uptime-kuma:2.5.5
    ports:
      - "127.0.0.1:18141:3001"
    environment:
      UPTIME_KUMA_DB_TYPE: sqlite
    volumes:
```

```
      - kuma_data:/app/data

volumes:
  kuma_data:
```

`run.sh` wipes the volume and retries. bcrypt cost 10 is a race, not a guarantee on the first try. Exit 2 is a miss. Fresh empty `user` table next.

PLAIN TEXT Copy

```
git clone https://github.com/abraxas/uptime-kuma-setup-toctou
cd uptime-kuma-setup-toctou/lab
./run.sh
```

The proof of concept is written for `127.0.0.1:18141`. Do not publish the port off loopback.

06 What the tree actually consumes

Connect sockets while the table is empty. Emit `setup` for two different usernames in the same poll. Read sqlite `user`. Sequential `setup` after that must be `initialized`. Extra user logs in, sets `disableAuth`. Reconnect auto-login identity is id 1.

A few other ways to lose without learning anything:

- One user row after concurrent setup. That is a race miss, not a patch. `run.sh` retries on a fresh volume.
- `UK-SETUP-TOCTOU-WITNESS` missing from `user`. UNIQUE collisions on the **same** name are expected. The leak is a **second** name.
- Sequential late setup accepted. Then the COUNT gate is gone entirely.

- Treating dashboard HTML 200 as SUCCESS. The oracle is two sqlite rows, then `disableAuth` auto-login as id 1.

- A reverse shell. Theatre.

Last lab run, trimmed:

PLAIN TEXT Copy

```
user-rows [('1', 'labadmin'), ('2', 'UK-SETUP-TOCTOU-WITNESS')]
race-count=2 witness=True
sequential-late-setup initialized=True
setSettings-disableAuth ack ok
reconnect autoLogin=True loginRequired=False
bonus-autologin-identity operator=True id1=labadmin auto_is_id1=True
SUCCESS uptime-kuma-setup-toctou
```

The client that produced it is on GitHub. I am not reprinting bcrypt rows, JWTs, or otpauth secrets.

07 What this is not

It is not "Uptime Kuma never rejects a second setup." After the race, sequential setup is `initialized`. The bug is two COUNT=0 handlers in flight. It is not an already-deployed instance with an admin. It is not GHSA-23q2-5gf8-gjpp (that one logs everyone out when auth is **re-enabled**). It is not RCE.

08 The fix

Serialize `setup` (transaction plus a one-row constraint, or a mutex around COUNT+INSERT). Do not treat `disableAuth` auto-login as `findOne` of the lowest id. Re-run the loopback client against a patched build: concurrent first-run `setup` must

not leave two `user` rows. Until then, do not expose first-run setup on a shared network.

I am not going to print a Socket.IO emit you can paste at someone else's `/setup`. The missing lock is the useful part. If you own the box, run the proof of concept against loopback.

09 References

- Proof of concept: abraxas/uptime-kuma-setup-toctou · uptime-kuma-setup-toctou-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-362 · CWE-367
- Tree: uptime-kuma 2.5.5 · `server.js` setup · `password-hash.js` · `knex_init_db.js`
- Nearby: GHSA-23q2-5gf8-gjpp (disableAuth re-enable logout; not this bug)
- SECURITY.md
- Product: Uptime Kuma

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