0-DAY NO CVE CRITICAL PUBLIC

Lago acceptInvite, cross-org account takeover

Lago v1.53.0 acceptInvite has no login. register_from_invite loads the user by invite email. If that account already has memberships, the posted password is ignored and a user-scoped JWT is issued. Inviter already has the token. No CVE yet.

NAME	TYPE
Lago acceptInvite, cross-org account takeover	0-day analysis
CVE	**CVE RISK**
n/a	critical
DISCLOSURE STATUS	**VENDOR**
public	Lago
AFFECTED	**PUBLISHED**
Lago through v1.53.0 email/password acceptInvite; unpublished. Google/Okta accept binds IdP email and is not this bug. Single-org self-host is weaker.	26 Sept 2026
UPDATED	**TAGS**
26 Sept 2026	0-day, lago, getlago, invite, cwe-287, cwe-306, authenticated, ato

TL;DR

Lab files and proof-of-concept here.

01 The invite token is a login

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

This is **Lago v1.53.0** (`ba292b6`), API lago-api `591ae9005` . No CVE yet. CWE-287 / CWE-306. 9.6 Critical (`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N`). You need an org you control (`organization:members:create`). Cloud signup is enough. `LAGO_DISABLE_SIGNUP` defaults to false. Not a shell. Not a random visitor with no account. Google/Okta invite accept is not this bug.

`Mutations::Invites::Accept` has no `AuthenticableApiUser` . Create/revoke do. Accept does not. `Invites::AcceptService` loads the pending invite **by token only**. Then `UsersService#register_from_invite` :

PLAIN TEXT Copy

```
user = User.find_or_initialize_by(email: invite.email)

if user.new_record?
  user.password = password
  user.save!
elsif user.memberships.active.none?
  user.update!(password:)
end
```

If that email already has an active membership, the posted password is discarded. `Utils::AuthToken.encode(user:)` still runs. The JWT is **user-scoped**. No org claim. The client sends `x-lago-organization` for whichever org that user already belongs to. Including the victim's.

The inviter already has the token. `Types::Invites::Object` exposes `token` on `createInvite`. You invite `cfo@victim` into a throwaway org, accept it yourself, and you are that person on every org that email already sits in. Invoices, API keys, Stripe, wallets. Mailbox never involved. Victim password unchanged.

I am not printing a GraphQL body you can paste at someone else's `/graphql`. The missing password check on an existing user is the useful part. Isolated lab. The stack is in the GitHub repo so you can run it at home.

02 What an attacker can do

You need an org you control. Cloud signup is enough (`LAGO_DISABLE_SIGNUP` defaults false). Invite a victim email, accept unauthenticated with a dummy password, then header-switch to Victim Corp. Invoices, API keys, Stripe, wallets.

Not a random visitor with no account. Not a password reset. Dummy login fails. Real victim password still works. Google/Okta accept binds IdP email and is not this bug. Single-org self-host with nobody else's email in the DB does not give you another merchant. Same database, two orgs, same email: it does.

Same tree as the Stripe one-time invoice IDOR. Different bug.

03 The lab (run this at home)

Source of truth is `lab/`. Image `getlago/lago:v1.53.0`. The Dockerfile is the image pin. Compose uses that image directly. Port **13000**. Bind it to loopback. No UI port.

DOCKERFILE Copy

```
# Loopback lab image pin for lago-invite-ato. Full stack: docker-compose.yml
FROM getlago/lago:v1.53.0
```

`docker-compose.yml`:

YAML Copy

```yaml
name: lago-invite-ato-lab

services:
  lago:
    image: getlago/lago:v1.53.0
    environment:
      LAGO_DISABLE_SIGNUP: "false"
      LAGO_DISABLE_PDF_GENERATION: "true"
      LAGO_DISABLE_SEGMENT: "true"
      LAGO_SIDEKIQ_WEB: "false"
      API_URL: http://127.0.0.1:13000
      LAGO_API_URL: http://127.0.0.1:13000
      LAGO_FRONT_URL: http://127.0.0.1:18080
    ports:
      - "127.0.0.1:13000:3000"
    volumes:
      - lago_invite_ato_data:/data
    healthcheck:
      test: ["CMD-SHELL", "curl -fsS http://127.0.0.1:3000/health || exit 1"]
      interval: 5s
      timeout: 5s
      retries: 60
      start_period: 40s

volumes:
  lago_invite_ato_data:
```

Signup stays on. Sidekiq Web stays off. That is not this bug. Two `registerUser` calls plant Victim Corp and Attacker Corp. Fresh emails each run so a dirty volume does not 422 `user_already_exists`.

PLAIN TEXT Copy

```
git clone https://github.com/abraxas/lago-invite-ato
cd lago-invite-ato/lab
./run.sh
```

`run.sh` waits for `/health`, then runs the client. The proof of concept is written for `127.0.0.1:13000`. Do not publish the port off loopback.

04 What the tree actually issues

HTTP is `POST /graphql`. Function names in the write-up are Ruby methods. `createInvite` is `AuthenticableApiUser` plus `RequiredOrganization` plus `REQUIRED_PERMISSION = "organization:members:create"`. That is an org admin. Then unauthenticated `acceptInvite`.

`AcceptService#call`:

PLAIN TEXT Copy

```
invite = args[:invite] || Invite.find_by(token: args[:token], status: :pending)
return result.not_found_failure!(resource: "invite") unless invite
# ...
result = UsersService.call(:register_from_invite, invite, args[:password])
result.token = generate_token(result.user, login_method: args[:login_method])
invite.recipient = result.membership
invite.mark_as_accepted!
```

No current-user. No mailbox proof. `generate_token` encodes the **found** user.

The password argument is required in GraphQL and unused when the user already has memberships. Lab posts a dummy. Login with that dummy fails. Login with the victim password still works. The ATO JWT is a session, not a password reset.

`GoogleService#accept_invite` is the control:

PLAIN TEXT Copy

```
unless google_oidc["email"] == invite.email
  return result.single_validation_failure!(error_code: "invite_email_mistmatch")
end
```

IdP accept binds the identity-provider email. Email/password accept does not bind the password of an existing user. Okta/Entra follow the Google pattern. This post is the password path.

`x-lago-organization` is how Lago switches tenants. A user-scoped JWT plus a victim org id is enough if that user is a member there. After accept, they are: the victim row already was, and the attacker org was just attached.

Cloud and Embedded share a user table by email. That is the interesting case. A single-org self-host with nobody else's email in the DB does not give you another merchant. Same database, two orgs, same email: it does.

05 The 200 that is a new user

The first client that looks at this will invite an unused address, accept it, and get a brand-new Lago user with the password they posted. That is the feature. The witness is the **existing** victim user id coming back.

A few other ways to lose without learning anything:

- Accepting with Google/Okta. Those check IdP email.
- Treating `createInvite` 401 without a JWT as the whole story. Create is authenticated. Accept is not.
- Looking only at the attacker org on `currentUser`. The tell is **Victim Corp** in the list, then `x-lago-organization` switching to it.
- Dummy password logging in. That would mean `register_from_invite` overwrote the hash. Lab asserts it did not.
- A reverse shell. Theatre. The witness is `LAGO-INVITE-ATO`: JWT user is the existing victim, orgs include both corps, real password still works, no inbox.

06 What I actually did

Treat the on-disk product as the spec. I read `Accept` (no `AuthenticableApiUser`), then `find_or_initialize_by(email:)`, then the memberships-present branch that skips the password, then `AuthToken.encode(user:)`. Proof of concept: lago-invite-ato-Abraxas-Labs.py.

Register two orgs, invite, accept unauthenticated, switch. Victim Corp. Attacker Corp. `createInvite` on the victim email. Unauthenticated `acceptInvite` with a dummy password. Returned user id matches the victim. `currentUser.organizations` lists both. Header switch returns Victim Corp. Dummy login fails. Real victim password still works. I am not reprinting JWTs.

Last lab run, trimmed:

PLAIN TEXT Copy

```
IOC victim_password_unchanged
IOC victim_real_password_still_works
SUCCESS LAGO-INVITE-ATO acceptInvite JWT is existing victim; lists Victim Corp; r
```

The client that produced it is on GitHub.

Wrong turns already recorded: accepting with Google/Okta; treating `createInvite` 401 without a JWT as the whole story (create is authenticated, accept is not); looking only at the attacker org on `currentUser` (the tell is **Victim Corp** in the list); dummy password logging in (that would mean the hash was overwritten - lab asserts it was not); a reverse shell. Theatre. No inbox.

07 What this is not

It is not "anyone on the internet logs in as any Lago user." You need an org you administer and the victim email. Cloud signup gives you the org. It is not IdP accept.

It is not a password reset. It is not RCE. It is a session as that person, across orgs, without their mailbox.

08 The fix

If the user already exists, require they authenticate as themselves (or prove mailbox / IdP) before attaching the membership. Do not mint a JWT for an existing user from unauthenticated accept. Stop returning invite `token` to the inviter if it is a capability. Re-run the loopback client against a patched build: `acceptInvite` with a dummy password must not return a JWT for the existing victim, and `currentUser` must not list Victim Corp.

I am not going to print a storefront GraphQL POST you can paste at someone else's Lago Cloud. The `find_or_initialize_by` plus skipped password is the useful part. If you own the box, run the proof of concept against loopback.

09 References

- Proof of concept: abraxas/lago-invite-ato · lago-invite-ato-Abraxas-Labs.py
- Lab: `lab/` · Dockerfile · docker-compose.yml · run.sh
- @abraxas_null · github.com/abraxas · abraxaslabs.tech · abraxas.null@proton.me
- CWE-287 · CWE-306
- v1.53.0: `accept.rb` · `accept_service.rb` · `users_service.rb` · `create.rb` · `object.rb` · `google_service.rb`
- tag v1.53.0 · lago · lago-api

- getlago.com/company/security
- Product: Lago · docs

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