0-DAY NO CVE HIGH PUBLIC

Lago Stripe one-time webhook, cross-org payment bypass

Lago v1.53.0 POST /webhooks/stripe/:org verifies the attacker org secret. metadata.payment_type=one-time then Invoice.find_by(id:) with no organization_id. handle_missing_payment is scoped. Dummy event. No CVE yet.

NAME Lago Stripe one-time webhook, cross-org payment bypass	TYPE 0-day analysis
CVE n/a	CVE RISK high
DISCLOSURE STATUS public	VENDOR Lago
AFFECTED Lago through v1.53.0 Stripe one-time webhook; unpublished. Shared database (Cloud / Embedded / multi-org). Adyen invoices are scoped. Single-org self-host is not this bug.	PUBLISHED 26 Sept 2026
UPDATED 26 Sept 2026	TAGS 0-day, lago, getlago, stripe, webhook, cwe-639, cwe-345, authenticated

TL;DR

Lab files and proof-of-concept here.

01 one-time skips the org check

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

This is **Lago v1.53.0** (`ba292b6`), API lago-api `591ae9005`. No CVE yet. CWE-639 / CWE-345. 7.7 High (`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:N`). Another tenant on the **same database** with Stripe connected. You need a victim invoice UUID (portal URL, email, PDF, leaked API). Not unauthenticated. Not a shell. Single-org self-host is not this bug: that signing secret already owns that org's payables.

Same tree as the acceptInvite ATO. Different bug.

`POST /webhooks/stripe/:organization_id` verifies `Stripe-Signature` with **that URL org's** webhook secret. Then the job hits `Invoices::Payments::StripeService#update_payment_status`:

PLAIN TEXT Copy

```
if !payment && stripe_payment.metadata[:payment_type] == "one-time"
  payment = create_payment(stripe_payment, amount_cents:)
end

unless payment
  handle_missing_payment(organization_id, stripe_payment)
  return result unless result.payment
  payment = result.payment
end
```

`create_payment` is the one-time branch. It loads the invoice with **no** org:

```
PLAIN TEXT
Copy
1 @invoice = invoice || Invoice.find_by(id: stripe_payment.metadata[:lago_invoice_i
```

`handle_missing_payment` is the control. It **is** scoped:

```
PLAIN TEXT
Copy
1 # NOTE: Invoice does not belong to this lago organization
2 #       It means the same Stripe secret key is used for multiple organizations
3 invoice = Invoice.find_by(id: stripe_payment.metadata[:lago_invoice_id], organiza
4 return result if invoice.nil?
```

Put `payment_type=one-time` on a signed `payment_intent.succeeded` aimed at **your** org URL, with `lago_invoice_id` of **their** invoice. The scoped path never runs. The victim row becomes `payment_status=succeeded`. Wallet top-up / payment-gated sub unlocks. Their Stripe dashboard has no matching charge. Yours might.

`Adyen` invoices already pass `organization_id:` into `find_by`. Cashfree and Flutterwave copy the unscoped Stripe `find_by(id:)`. This lab is Stripe.

I am not printing a signed webhook JSON you can paste at someone else's `/webhooks/stripe/`. The missing `organization_id:` on the one-time path is the useful part. Isolated lab. Dummy event. No live Stripe.

02 What an attacker can do

Sign a `payment_intent.succeeded` as **their** org, put `payment_type=one-time` and `lago_invoice_id` of **your** invoice. The scoped path never runs. The victim row becomes `payment_status=succeeded`. Wallet top-up / payment-gated sub unlocks.

Not unauthenticated. Not a shell. Single-org self-host is not this bug: that signing secret already owns that org's payables. You need a victim invoice UUID (portal URL,

email, PDF, leaked API) and a victim `stripe_customer` row. Dummy event. No live Stripe.

Same tree as the acceptInvite ATO. 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 **13001**. Bind it to loopback. Invite ATO used **13000**. This one does not share a volume with that lab.

DOCKERFILE Copy

```dockerfile
# Loopback lab image pin for lago-stripe-invoice-idor. Full stack: docker-compose
FROM getlago/lago:v1.53.0
```

`docker-compose.yml`:

YAML Copy

```yaml
name: lago-stripe-invoice-idor

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://127.0.0.1:13001
      LAGO_API_URL: http://127.0.0.1:13001
      LAGO_FRONT_URL: http://127.0.0.1:18081
    ports:
      - "127.0.0.1:13001:3000"
    volumes:
      - lago_stripe_idor_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_stripe_idor_data:
```

Two `registerUser` calls plant Victim Corp and Attacker Corp. Victim REST key creates an add-on, a customer, and an unpaid invoice with `skip_psp`. That flag is a lab fixture so you do not need a live PSP to mint the row. `seed_stripe.rb` (the client also inlines it) plants Stripe providers and a `stripe_customer` on the victim. Dummy `webhook_secret`. Dummy `sk_test_lab_*`. Not a production key.

PLAIN TEXT Copy

```
git clone https://github.com/abraxas/lago-stripe-invoice-idor
cd lago-stripe-invoice-idor/lab
./run.sh
```

`run.sh` waits for `/health`, then runs the client. The proof of concept is written for `127.0.0.1:13001`. Do not publish the port off loopback.

04 What the tree actually looks up

`WebhooksController#stripe` is the HTTP door:

PLAIN TEXT Copy

```
result = InboundWebhooks::CreateService.call(
  organization_id: params[:organization_id],
  webhook_source: :stripe,
  payload: request.body.read,
  signature: request.headers["HTTP_STRIPE_SIGNATURE"],
  event_type: params[:type]
```

```
)
return head(:bad_request) unless result.success?
head(:ok)
```

Signature is real. Org in the URL is the **attacker**. The invoice id in metadata is whoever you named. Function names in the write-up are Ruby methods. The route is `/webhooks/stripe/:organization_id`.

Once `create_payment` has an invoice, it attaches a `Payment` to **that invoice's** organization, not the webhook URL's. `update_invoice_payment_status` then sets `payment_status: succeeded` and `total_paid_amount_cents`. Lab catalog **1000** cents.

The same Stripe secret used across orgs is what the comment on `handle_missing_payment` was afraid of. The one-time branch forgot to be afraid.

Need a victim `stripe_customer`. `create_payment` reads `customer.stripe_customer.id`. No Stripe customer, the one-time path throws instead of paying. That is a precondition, not the bug. Cloud customers who pay with Stripe already have the row.

05 The 200 that stayed pending

The first client that looks at this will POST a `payment_intent.succeeded` without `payment_type`. `handle_missing_payment` looks up the invoice **and** `organization_id`. Attacker org, victim invoice. `nil`. Pending. That is the control. Lab asserts it.

A few other ways to lose without learning anything:

- Signing with the **victim** webhook secret. Then you already own that merchant's payables.
- Single-org self-host. Same.

- Invoice UUID from the attacker org. In-org payment. Not the bug.
- No `stripe_customer` on the victim. Exception, not `succeeded`.
- Treating webhook HTTP 200 as the oracle. The job is async. Poll `GET /api/v1/invoices/:id` until `payment_status`.
- A reverse shell. Theatre. The witness is victim `payment_status=succeeded` and `total_paid_amount_cents=1000` after a dummy event, with the control still pending.

06 What I actually did

Treat the on-disk product as the spec. I read `update_payment_status`, then the one-time `create_payment` `find_by(id:)`, then `handle_missing_payment`'s `organization_id:`. Proof of concept: lago-stripe-invoice-idor-Abraxas-Labs.py.

Two orgs, unpaid invoice, control, one-time. Victim add-on 1000 cents. Unpaid invoice `skip_psp`. Signed control event, no `payment_type`: still pending. Signed event with `payment_type=one-time`: `succeeded`, `paid=1000`. Dummy Stripe. I am not reprinting org UUIDs or webhook secrets.

Last lab run, trimmed:

PLAIN TEXT Copy

```
IOC payment_status_before=pending
IOC control webhook (no payment_type) payment_status_after_control=pending
IOC attack webhook payment_type=one-time
IOC poll payment_status=succeeded paid=1000
SUCCESS Lago Stripe one-time webhook unscoped invoice
```

The client that produced it is on GitHub.

Wrong turns already recorded: signing with the **victim** webhook secret (then you already own that merchant's payables); single-org self-host; invoice UUID from the attacker org (in-org payment); no `stripe_customer` on the victim (exception, not `succeeded`); treating webhook HTTP 200 as the oracle (the job is async - poll `GET /api/v1/invoices/:id`); a reverse shell. Theatre.

07 What this is not

It is not "unauthenticated anyone marks invoices paid." You sign as an org that connected Stripe. It is not a race. It is not acceptInvite. It is not Adyen invoices. It is not a live charge on the victim merchant. It is a Lago row that says paid.

08 The fix

`Invoice.find_by(id:, organization_id:)` on the one-time path, the way `handle_missing_payment` and Adyen invoices already do. Same for Cashfree/Flutterwave. Re-run the loopback client against a patched build: the one-time event must leave the victim invoice pending (or 404), same as the control.

I am not going to print a Stripe-signed body you can paste at someone else's Lago Cloud. The unscoped `find_by(id:)` is the useful part. If you own the box, run the proof of concept against loopback.

09 References

- Proof of concept: abraxas/lago-stripe-invoice-idor · lago-stripe-invoice-idor-Abraxas-Labs.py
- Lab: `lab/` · Dockerfile · docker-compose.yml · run.sh · seed_stripe.rb

- Same product: Lago acceptInvite, cross-org account takeover
- @abraxas_null · github.com/abraxas · abraxaslabs.tech · abraxas.null@proton.me
- CWE-639 · CWE-345
- v1.53.0: `webhooks_controller.rb` · `stripe_service.rb` · `adyen_service.rb` · `cashfree_service.rb` · `flutterwave_service.rb` · `routes.rb`
- tag v1.53.0 · lago · lago-api
- getlago.com/company/security
- Product: Lago · docs · Stripe webhooks

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