Abraxas Labs
ABRAXAS

DARK LIGHT

Research / hyperswitch-unsigned-webhook

0-DAY NO CVE HIGH PUBLIC

Hyperswitch unsigned Worldpayxml webhook, unauthenticated payment bypass

Hyperswitch 2026.09.21.0 POST /webhooks/{merchant_id}/worldpayxml. Worldpayxml verify_webhook_source returns Ok(true). incoming.rs HandleResponse without PSync. lastEvent SETTLED maps to Charged. Dummy XML. No CVE yet.

NAME Hyperswitch unsigned Worldpayxml webhook, unauthenticated payment bypass	TYPE 0-day analysis
CVE n/a	CVE RISK high
DISCLOSURE STATUS public	VENDOR Juspay
AFFECTED Hyperswitch through 2026.09.21.0 Worldpayxml inbound webhooks; unpublished. BitPay/Shift4 inherit NoAlgorithm. Needs merchant_id and orderCode.	PUBLISHED 28 Sept 2026
UPDATED 28 Sept 2026	TAGS 0-day, hyperswitch, juspay, worldpayxml, webhook, cwe-345, cwe-306, unauthenticated

TL;DR

Lab files and proof-of-concept here.

01 Ok(true) is not mTLS

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

This is **Hyperswitch 2026.09.21.0** (Juspay). Lab image `hyperswitch-router:standalone` v1.127.0. No CVE yet. CWE-345 / CWE-306. 7.5 High (`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N`). Unauthenticated `POST /webhooks/{merchant_id}/worldpayxml`. You need the merchant id in the path and a connector `orderCode` that matches `connector_transaction_id`. Not a shell. Not a Worldpay capture.

`Worldpayxml::verify_webhook_source`:

PLAIN TEXT Copy

```
async fn verify_webhook_source(
    &self,
    _request: &webhooks::IncomingWebhookRequestDetails<'_>,
    // ...
) → CustomResult<bool, errors::ConnectorError> {
    // Bypass Source Verification since it is done via MTLS
    Ok(true)
}
```

There is no mTLS on this route. The function ignores the body and the headers. `incoming.rs` then trusts `source_verified`:

PLAIN TEXT Copy

```
let consume_or_trigger_flow = if source_verified {
    // ...
    payments::CallConnectorAction::HandleResponse {
        resource_object,
        event_type: Some(event_type.into()),
    }
} else {
    payments::CallConnectorAction::Trigger
};
```

Verified means **HandleResponse**. No PSync. Unsigned SETTLED XML is "verified." `get_payment_webhook_event` maps `LastEvent::Settled` to `PaymentIntentSuccess`. The intent goes **Charged**. Lab: `status=succeeded`. Dummy XML. No PSP.

The default `IncomingWebhook` algorithm is `NoAlgorithm`. `verify_signature` returns `Ok(true)`. BitPay and Shift4 inherit that. Worldpayxml does not even get that far. It short-circuits.

I am not printing a `paymentService` notify body you can paste at someone else's `/webhooks/{merchant_id}/worldpayxml`. The `Ok(true)` is the useful part. Isolated lab.

02 What an attacker can do

Need `merchant_id` (in every Hyperswitch dashboard URL) and `orderCode` equal to the attempt's `connector_transaction_id`. POST `/webhooks/{merchant_id}/worldpayxml` unsigned SETTLED. If `source_verified` were false, incoming would `Trigger` and PSync the connector. Worldpayxml never takes that path. Intent goes **Charged**. Lab: `status=succeeded`. Dummy XML. No PSP.

Guessing payment ids is not the bug. Forging SETTLED for an id you already saw is. Not a shell. Not a live Worldpay settlement.

Same product, different bug: payout confirm unbound client_secret.

03 The lab (run this at home)

Source of truth is `lab/`. Router image `docker.juspay.io/juspaydotin/hyperswitch-router:standalone`. Compose also runs `postgres:16`, `redis:7`, and `superposition-demo:0.113.0`. The Dockerfile pins `debian:trixie-slim` for the Diesel migrator. Compose uses that image for `migration_runner` and the Juspay router for the API. Port **18082**. Bind it to loopback.

DOCKERFILE Copy

```
# Loopback lab image pin for hyperswitch-unsigned-webhook. Full stack: docker-com
FROM docker.io/debian:trixie-slim
```

`run.sh` clones Hyperswitch tag **2026.09.21.0** into `lab/hyperswitch-src` for config and migrations. Then compose.

`docker-compose.yml` (truncated to the router; full file on GitHub):

YAML Copy

```yaml
name: hyperswitch-unsigned-webhook

services:
  pg:
    image: docker.io/postgres:16
    environment:
      POSTGRES_USER: db_user
      POSTGRES_PASSWORD: db_pass
      POSTGRES_DB: hyperswitch_db

  redis-standalone:
    image: docker.io/redis:7

  hyperswitch-server:
```

```
    image: docker.juspay.io/juspaydotin/hyperswitch-router:standalone
    command: /local/bin/router -f /local/config/docker_compose.toml
    ports:
      - "127.0.0.1:18082:8080"
    volumes:
      - ./hyperswitch-src/config:/local/config:ro
      - ./files:/local/bin/files
```

The client creates a merchant, an API key, a worldpayxml MCA, and a payment with `confirm=false`. It then stamps `connector_transaction_id` so `orderCode` can match. That stamp is a lab fixture. A live attempt already has the connector id. The webhook does not.

PLAIN TEXT Copy

```
git clone https://github.com/abraxas/hyperswitch-unsigned-webhook
cd hyperswitch-unsigned-webhook/lab
./run.sh
```

The proof of concept is written for `127.0.0.1:18082`. Do not publish the port off loopback.

04 What the tree actually consumes

HTTP is `POST /webhooks/{merchant_id}/worldpayxml`. `MerchantIdAuth` from the path. Function names in the write-up are Rust methods. Content-Type is XML.

If `source_verified` is false, incoming uses `Trigger` and PSyncs the connector. That is the honest path. Worldpayxml never takes it. The comment says mTLS. The lab POST has no client cert. HTTP 200. Intent **succeeded**.

Same class, not this lab: connectors that leave the default `NoAlgorithm` / empty signature / empty message. `NoAlgorithm::verify_signature` is `Ok(true)`. BitPay, Shift4. Worldpayxml is louder because it does not pretend to hash.

Need `merchant_id` (in every Hyperswitch dashboard URL) and `orderCode` equal to the attempt's `connector_transaction_id`. Guessing payment ids is not the bug. Forging SETTLED for an id you already saw is.

05 The 200 that stayed requires_capture

The first client that looks at this will POST the webhook against `requires_confirmation`. Incoming will not Charged a payment that has not left confirmation. Stamp the attempt, or wait until `requires_capture` / `processing`. Lab does the stamp.

A few other ways to lose without learning anything:

- Signed HMAC on a connector that actually verifies. This path does not.
- Treating webhook HTTP 200 as the oracle. Poll `GET /payments/{id}` until `status=succeeded`.
- `orderCode` that is not `connector_transaction_id`. Reference miss.
- A reverse shell. Theatre. The witness is payment **succeeded** after unsigned SETTLED, with no Worldpay capture.

06 What I actually did

Treat the on-disk product as the spec. I read `Ok(true)`, then `HandleResponse` vs `Trigger`, then `Settled → PaymentIntentSuccess`. Proof of concept: hyperswitch-unsigned-webhook-Abraxas-Labs.py.

Merchant, MCA, payment, unsigned SETTLED, retrieve. Dummy XML. I am not reprinting `orderCode` values.

Last lab run, trimmed:

```
IOC status_before=requires_confirmation
IOC webhook status=200 unsigned lastEvent=SETTLED
IOC poll status=succeeded
SUCCESS Hyperswitch unsigned Worldpayxml webhook Charged
```

The client that produced it is on GitHub.

Wrong turns already recorded: signed HMAC on a connector that actually verifies (this path does not); treating webhook HTTP 200 as the oracle (poll `GET /payments/{id}` until `status=succeeded`); `orderCode` that is not `connector_transaction_id`; a reverse shell. Theatre.

07 What this is not

It is not "Hyperswitch never PSyncs." Unverified connectors Trigger. Worldpayxml is marked verified with no check. It is not a live Worldpay settlement. It is not RCE. It is a payment row that says succeeded.

08 The fix

Verify the Worldpay notify (signature / mTLS that the route actually terminates), and do not `HandleResponse` on a connector that returned `Ok(true)` from a no-op. Re-run the loopback client against a patched build: unsigned SETTLED must not move the intent to succeeded.

I am not going to print a notify XML you can paste at someone else's Hyperswitch. The `Ok(true)` is the useful part. If you own the box, run the proof of concept against loopback.

Same product, different bug: payout confirm unbound client_secret.

09 References

- Proof of concept: abraxas/hyperswitch-unsigned-webhook · hyperswitch-unsigned-webhook-Abraxas-Labs.py
- Lab: `lab/` · Dockerfile · docker-compose.yml · run.sh
- @abraxas_null · github.com/abraxas · abraxaslabs.tech · abraxas.null@proton.me
- CWE-345 · CWE-306
- 2026.09.21.0: `worldpayxml.rs` · `transformers.rs` · `incoming.rs` · `webhooks.rs` · `crypto.rs`
- tag 2026.09.21.0 · juspay/hyperswitch
- VDP
- Product: Hyperswitch · docs

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