



---

LITHICTESTING AND GALILEOTESTING

# Sandbox testing proxies

Why stage needs them, how they are built, and how to build one

## WHY

# The sandbox cannot describe the scenario



### **Simulate cannot set terminal type**

Lithic's simulate request takes an MCC, but no terminal type, entry mode or terminal id.



### **Sandbox webhooks are generic**

In real captures, the ASA body carried terminal PHONE and entry mode MANUAL.



### **Production reads those fields**

The transaction type parser needs them to tell an ATM withdrawal from a POS purchase.

Before the relay patched POS data, simulated transactions were typed as **Visa money transfers**.

## WHY

# One simulated ATM withdrawal

### SANDBOX SENDS

```
"pos": {  
  "terminal": {  
    "type": "PHONE",  
    "attended": false  
  },  
  "entry_mode": {  
    "pan": "MANUAL",  
    "cardholder": "UNKNOWN"  
  }  
}
```

**Parsed as: Visa money transfer**

### RELAY SENDS

```
"pos": {  
  "terminal": {  
    "type": "ATM",  
    "attended": false  
  },  
  "entry_mode": {  
    "pan": "ICC",  
    "cardholder": "PRESENT"  
  }  
}
```

**Parsed as: ATM withdrawal**

Left: abridged real sandbox captures (non-ATM MCCs). Right: abridged relay output for MCC 6011, as asserted by its unit tests.

## WHY

# Why a separate pair of services

### Test code stays out of the real service

The overrides live in their own deployable projects, outside the production service.

### A real call, not a mock

The real service still receives a real request, re-signed for Lithic, through its real controllers.

### Easy to switch back

Pointing the vendor sandbox at the real service again is a registration change, not a code change.

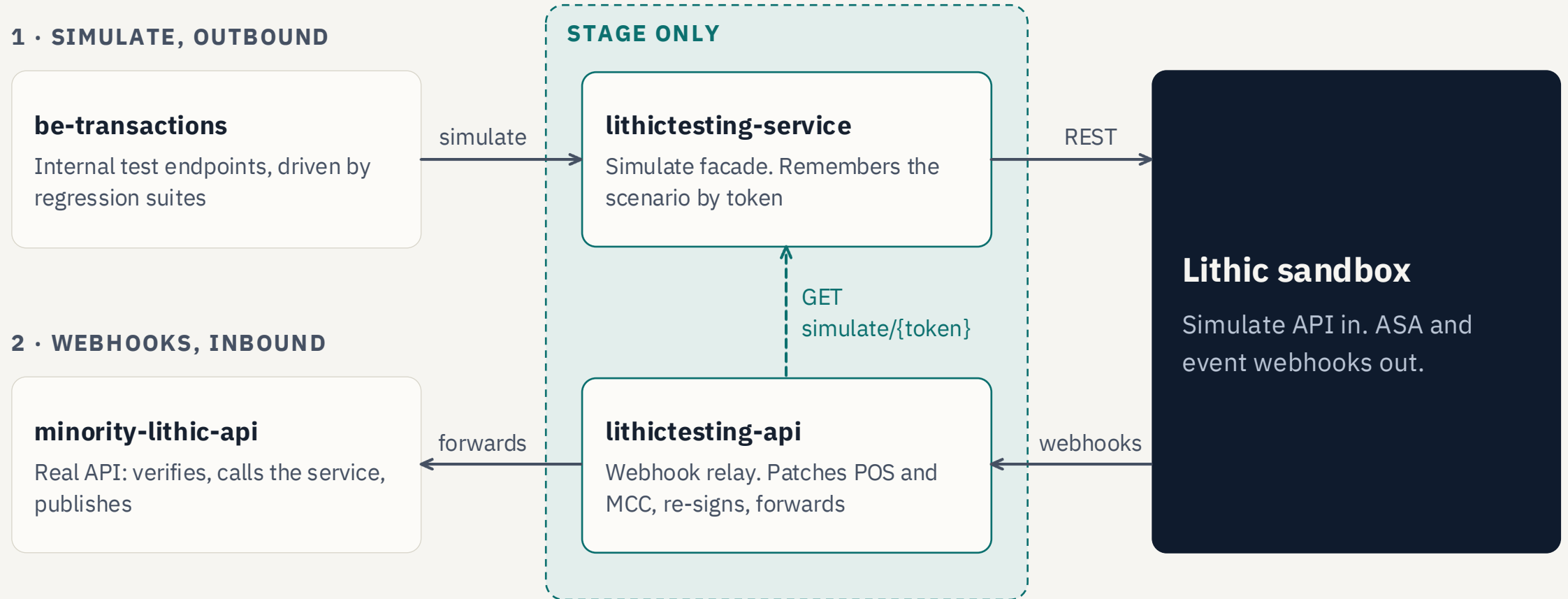
### THE ORIGINAL TICKET, MIB-6234 (2022)

“Build a proxy to insert merchant and transaction type and more on the api call ... use only our API code.”

## ARCHITECTURE

# A simulate service and a webhook relay

### 1 • SIMULATE, OUTBOUND



## ARCHITECTURE

# Inbound: patch, re-sign, forward

1

### Receive

The same two routes as the real API: Authorization (ASA) and Event.

2

### Patch

ASA: terminal type from the MCC. Event: stored scenario sets type, id and MCC.

3

### Re-sign

HMAC with the webhook secret of that card program.

4

### Forward

The real service verifies and processes. Its answer goes back to Lithic.

## HOW LITHIC FINDS THE RELAY

We register the relay's URL with Lithic as the ASA responder and event endpoint. Nothing intercepts the sandbox on the network.

## WHY RE-SIGN

The body is re-serialised and often patched, so Lithic's signature no longer matches. The real service checks it.

## ARCHITECTURE

# Outbound: simulate, then remember

- 1 be-transactions calls simulate/authorize with a card token, amount and descriptor. No PAN.
- 2 Look up the card and its program: debit, credit or Rain Crypto.
- 3 Fetch the PAN through the CDE accounts client, where the call needs it.
- 4 Call Lithic's sandbox simulate endpoint.
- 5 Save the request model, keyed by the Lithic transaction token.

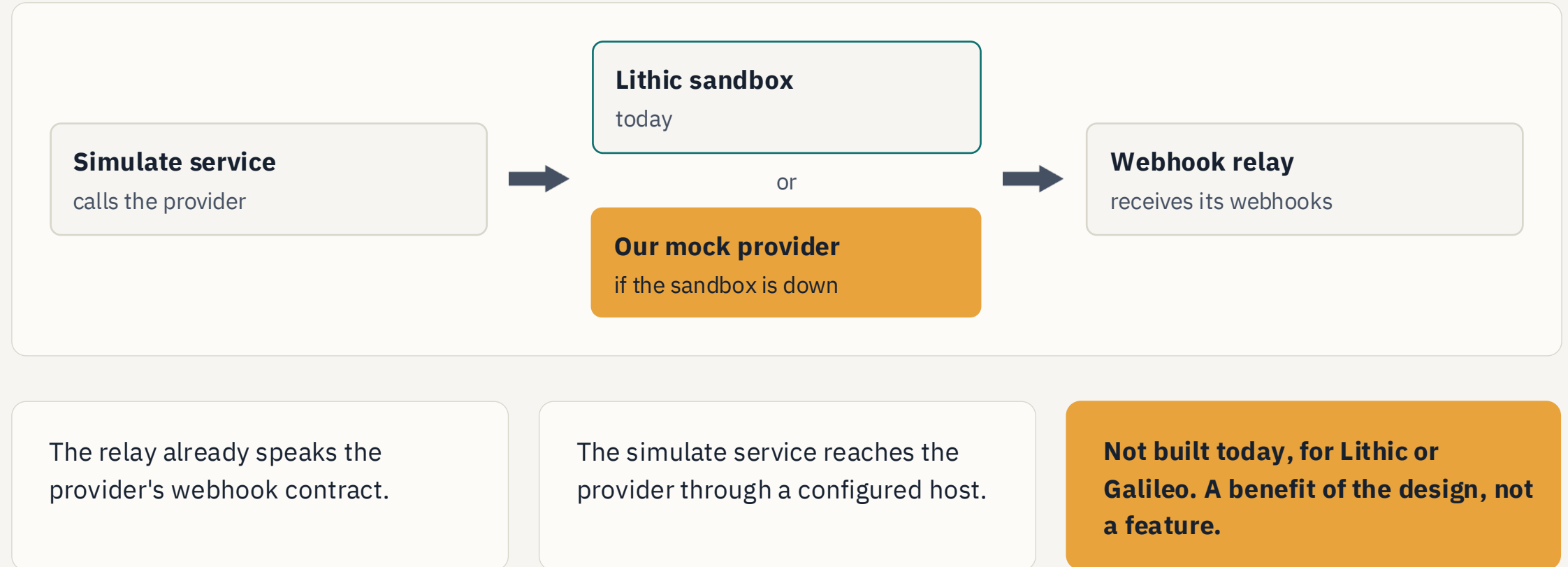
### dbo.SimulateRequest

| Column  | Holds                                        |
|---------|----------------------------------------------|
| Token   | Lithic transaction token (PK)                |
| Type    | Authorization, advice, credit advice, return |
| Payload | Request model JSON, no PAN                   |
| Created | UTC timestamp                                |

Written for four of the seven simulate calls. Clearing, void and return reversal save nothing.

## ARCHITECTURE

# Bonus: the seams can host a mock provider



## GUARDRAILS

# Stage only, enforced in layers

| Layer             | Rule                                                                        |
|-------------------|-----------------------------------------------------------------------------|
| 1. CI workflow    | Manual workflow_dispatch only, with stageOnly: true to skip the prod deploy |
| 2. Argo CD        | The bank-lithictesting app is registered for env: [stage]                   |
| 3. be-deployments | No prod-values.yaml: only stage-values.yaml (and a stale dev one) exist     |
| 4. OpenTofu       | Identities, database and key vault use count 0 when environment is prod     |

LINEAGE

# Lithic and Galileo: same idea, two shapes

| Aspect            | LithicTesting                  | GalileoTesting                   |
|-------------------|--------------------------------|----------------------------------|
| Deployables       | 2: relay API, simulate service | 3: Auth.Api, Events.Api, Service |
| Scenario store    | Read through the service       | SQL table read by the proxies    |
| Correlation       | Lithic transaction token       | Temp id in the merchant name     |
| Webhook signature | Re-signed per card program     | JWT header copied as is          |
| Status            | Live on stage                  | Retired, Argo app removed        |

2026-04-22

Traffic moves to Lithic (tickets say 04-23)

2026-06-11

Galileo webhook controllers unregistered

2026-08-04

Galileo services removed from prod

2026-09-29

GalileoTesting Argo app removed

HOW TO BUILD ONE

# Six projects in LithicTesting

| Project          | Role                                 |
|------------------|--------------------------------------|
| Api.Contract     | Mirrors the real controllers         |
| Api              | Webhook relay, public host           |
| Service.Contract | The /simulate routes                 |
| Service          | Simulate facade                      |
| Db               | SimulateRequest table                |
| Test             | Tests with the real contracts mocked |

```
LithicTestingApiModule, abridged

builder.ConfigureHttpClient(Client).Build();
builder.RegisterRestClientFor<IAuthController>(
    contentType: Client, httpClientName: Client);
builder.RegisterController<AuthTestController,
    IAuthTestController>();
builder.RegisterContentFormat<
    LithicTestingRequestFactory,
    MajorityResponseFactory>(Client);
```

Mirrored controller in, real contract out, a custom request factory in between.

## HOW TO BUILD ONE

# Mirror, patch, forward, re-sign

### 1 • Test controller: patch, then call the real one

```
public async Task<AsaResponseModel> Authorize(string cardProgramType, AsaRequestModel request)
{
    request.Pos = request.Pos.FixBasedOnSimulateRequest(request.Merchant?.Mcc);
    return await _authController.Authorize(cardProgramType, request);
}
```

### 2 • Request factory: sign what you actually send

```
var secret = (info.RelativeUrl.EndsWith("/event", true, CultureInfo.InvariantCulture)
    ? _credentials.EventWebhookKey(programType)
    : _credentials.AuthStreamWebhookKey(programType))
?? throw new ArgumentOutOfRangeException(...);
var signature = SignatureHelper.GetSignature(secret, msgId, msgTimestamp, messageBody);
```

## HOW TO BUILD ONE

# Rollout checklist, in dependency order

| # | Where                   | Change                                                                       |
|---|-------------------------|------------------------------------------------------------------------------|
| 1 | be-<vendor>             | Testing projects in the vendor solution, plus a stage-only workflow          |
| 2 | platform-infrastructure | OpenTofu: identities, SQL database, key vault, all count 0 in prod           |
| 3 | be-configuration        | Stage app-settings: key vault names, plus connection strings for the service |
| 4 | be-deployments          | stage-values.yaml per app, and no prod-values.yaml                           |
| 5 | platform-infrastructure | Argo app for stage, network policies, Cloudflare DNS record                  |
| 6 | Lithic sandbox API      | Register the relay URL: ASA responder endpoint and event subscription        |
| 7 | Pipeline                | Run the workflow by hand: image build, values PR, Argo sync                  |

## HOW TO BUILD ONE

# When to build another pair

### Build one when

- Stage talks to the vendor sandbox and prod to the real API.
- The sandbox cannot send fields that production code branches on.
- Fixing it inside the real service would put test code in prod paths.
- Other stage services need an easy way to drive the vendor.
- Need to switch between mocked and real sandbox

### Skip it when

- The sandbox payload already matches production.
- The sandbox response pace and quality is enough, so we don't need to mock it

## SUMMARY

# Takeaways

---

- 1 The sandbox cannot say what we mean, so the pair says it for us.
- 2 The relay patches, re-signs and forwards. The service drives the simulator and remembers the scenario.
- 3 Stage only and narrow: simulated transactions arrive as chip POS or ATM, not every card scenario.