

ES-2026-0114 — Security Assessment (Sample)

Engagement ES-2026-0114 — ACME Commerce

PRO engagement. Client, hostnames and payload data sanitized as agreed in the authorization letter. This is the published sample report: a real engagement's structure and register, with an invented client. Every section below appears in the report you receive.

Engagement at a glance

Scope	shop.example (storefront + checkout) and api.shop.example (v1 REST API). Staff admin at admin.shop.example in scope for authenticated testing. Marketing site and the PSP's hosted card iframe explicitly out of scope.
Roles	Unauthenticated · guest checkout · customer · staff (fulfilment) · staff (admin) — five perspectives, two separate customer accounts for tenancy checks
Test window	2026-01-12 → 2026-01-16, four days
Findings	8 (0 critical · 1 high · 3 medium · 2 low · 2 info)
Re-test	Day 19 — 6 closed, 2 accepted as risk by the client

Executive summary

ACME Commerce is **soundly built, with one finding that costs you money and three worth a sprint**. Nothing we found exposes cardholder data, and nothing gives an attacker a path to full compromise. The headline items:

- HIGH** — a customer can stack a fixed-amount discount code until an order total falls to near zero and check out at that price (ES-0114-01). This is a revenue problem, not a data-breach problem, but it is exploitable today by anyone with an account and it does not require any tooling.
- MEDIUM** — guest order tracking can be walked with a sequential order number and a common surname, returning the delivery address and line items (ES-0114-02).

3. **MEDIUM** — demoting a staff member does not invalidate their live session; they keep admin capability for up to eight hours (ES-0114-03).

The things that most often go wrong in a store of this size did not go wrong here. Card data never reaches your origin — the PSP iframe is correctly integrated and we could not make the checkout page touch a PAN. The admin panel is IP-restricted and enforces MFA. We put 340 injection payloads through every parameter we could reach and found no SQL injection, no command injection and no server-side template injection. Password change requires the current password, and profile updates silently ignore fields the user is not allowed to set.

The two low findings and both informational findings are hygiene. We would fix the two lows in a backlog sprint and would not lose sleep over the informational pair — and we have said so below rather than padding the count.

Findings

Ordered by severity, highest first. Every finding was reproduced by hand before it was written down.

HIGH — Fixed-amount discount codes stack without limit

ES-0114-01 · `POST /api/v1/cart/coupon`

Impact

A fixed-amount discount code (e.g. `WELCOME15`, $-\$15.00$) can be applied to the same cart repeatedly. The server validates that the code exists and is in date, but never checks whether it is already attached to the cart, and never floors the resulting total at the order's cost basis. Twelve applications of a \$15 code took a \$172.40 cart to \$0.01 and the order completed, was charged, and entered fulfilment as a normal paid order.

This needs no tooling and no special access — a signed-in customer clicking “apply” twelve times reproduces it. It is a direct revenue loss on every order an aware customer places, and because the order looks legitimate downstream, nothing in fulfilment or reconciliation flags it.

The fix

Enforce the constraint server-side in the cart service, not in the checkout controller: reject a coupon already present on the cart, cap total discount at the order subtotal, and floor the payable amount at a configured minimum. Add a regression test that applies the same code twice and asserts the second call returns `409`.

Evidence

```
POST /api/v1/cart/coupon      {"code": "WELCOME15"}
→ 200 {"subtotal": "172.40", "discount": "15.00", "total": "157.40"}
```

... same request × 11 ...

→ 200 {"subtotal":"172.40","discount":"172.39","total":"0.01"}

POST /api/v1/checkout → 201 {"order":"ACME-80412","status":"paid"}

→ order ACME-80412 charged \$0.01 and moved to fulfilment.

→ proof only: test account, test card, order cancelled with the client at 16:05 UTC.

OWASP: A04 Insecure Design **Fix effort:** Small **Verified by hand:** yes

MEDIUM — Guest order tracking is walkable with an order number and a surname

ES-0114-02 · GET /orders/track?order={n}&surname={s}

Impact

Guest order lookup takes a sequential eight-digit order number and a surname. Order numbers increment predictably (we observed ACME-80397 through ACME-80412 across our own test orders, a gap of 15 over two days), and the surname check is case-insensitive with no rate limit beyond a 60-request-per-minute edge throttle and no lockout. Pairing a guessed order number with the fifty most common surnames in the store's market returns, on a hit, the delivery address, the line items and the order total.

Reach is bounded — one order per successful guess, no account access, no payment data — which is why this is Medium rather than High. But it is an unauthenticated read of customer addresses, and the throttle is a speed bump rather than a control.

The fix

Stop treating the surname as a secret. Issue a signed, single-order lookup token in the confirmation email and make that the only way into /orders/track . If you keep the form, add per-IP and per-order-number lockout after five failed surname attempts and switch order numbers to a non-sequential identifier.

Evidence

GET /orders/track?order=ACME-80404&surname=okafor → 404 "not found"

GET /orders/track?order=ACME-80404&surname=REDACTED → 200

```
{"order": "ACME-80404", "placed": "2026-01-13",  
  "ship_to": {"line1": "REDACTED", "city": "REDACTED", "post": "REDACTED"},  
  "items": [{"sku": "REDACTED", "qty": 2}], "total": "64.00"}
```

→ 300 lookups in 5m12s from one address: no 429, no lockout, no alert.
→ we stopped at 300 and confirmed one hit against an order we had placed ourselves.

OWASP: A01 Broken Access Control **Fix effort:** Medium **Verified by hand:** yes

MEDIUM — Staff role changes do not take effect until the session expires

ES-0114-03 · `PATCH /api/v1/staff/{id}` → `admin.shop.example`

Impact

Staff authorization is read from a claim inside an eight-hour JWT and never re-checked against the database. Demoting a staff member from `admin` to `fulfilment` — or deactivating them entirely — updates the record but leaves their issued token fully privileged until it expires. In our test the demoted account continued to create discount codes and read the customer export for the remainder of the window.

The realistic case is an offboarding: the moment you most need a role change to take effect is the moment it does not. Medium rather than High because it requires an already-trusted account and a window measured in hours.

The fix

Check role and account status server-side on every privileged request rather than trusting the claim, or keep the claim and maintain a revocation list that the auth middleware consults. Shorten the access-token lifetime and issue refresh tokens so a role change is picked up within minutes.

Evidence

```
PATCH /api/v1/staff/4471 {"role":"fulfilment"} → 200 {"role":"fulfilment"}
GET /api/v1/staff/4471 → 200 {"role":"fulfilment"}
```

same JWT, issued while the account was still admin:

```
POST /api/v1/admin/discounts {"code":"ES-PR00F"} → 201 created
```

```
GET /api/v1/admin/customers/export → 200 (14,802 rows)
```

→ privileged for 7h41m after demotion.

→ proof only: the discount code was created inactive and deleted at 15:20 UTC;
the export was requested with `Range: bytes=0-2047` and not retained.

OWASP: A01 Broken Access Control **Fix effort:** Medium **Verified by hand:** yes

MEDIUM — Payment webhook accepts unsigned callbacks

ES-0114-04 · `POST /hooks/payments`

Impact

The payment provider signs every callback with an HMAC in the `X-Signature` header. Your handler does not verify it — it reads the body and updates the order. The only thing standing between a forged callback and an order marked `paid` is an IP allowlist covering the provider's published egress ranges.

Those ranges are shared, documented infrastructure, so the control is real but weak: an attacker needs an origin inside them rather than a secret. We confirmed the missing verification by replaying a valid callback with a mutated body and a deliberately wrong signature, from an allowlisted test origin the client provisioned for us. Medium, because the IP allowlist genuinely raises the bar.

The fix

Verify the HMAC against the raw request body before parsing it, using a constant-time comparison, and reject on mismatch. Keep the IP allowlist as defence in depth rather than as the control. Treat `order.amount_paid` from the callback as untrusted and reconcile it against the charge you created.

Evidence

```
POST /hooks/payments
X-Signature:
sha256=0000000000000000000000000000000000000000000000000000000000000000
{"event":"charge.succeeded","order":"ACME-80409","amount":"0.00"}

→ 200 {"ok":true}
→ order ACME-80409 transitioned unpaid → paid on a signature of all zeroes.
→ proof only: test order, reverted with the client at 11:40 UTC.
```

OWASP: A08 Software & Data Integrity Failures **Fix effort:** Trivial **Verified by hand:** yes

LOW — Session cookie sets Secure and HttpOnly but not SameSite

ES-0114-05 · Set-Cookie: `acme_sid`

Impact

The session cookie is correctly marked `Secure` and `HttpOnly`, so it is not readable by script and not sent in the clear. `SameSite` is absent, which means browsers fall back to their own default — `Lax` in current Chrome and Firefox, but not guaranteed and not uniform across embedded webviews.

Combined with the absence of a CSRF token on two state-changing form posts, this leaves a narrow cross-site request forgery window on older and embedded clients.

The fix

Set `SameSite=Lax` explicitly on the session cookie, and add a synchroniser token to `POST /account/address` and `POST /account/preferences` — the two form posts that rely on cookie auth alone.

Evidence

```
HTTP/2 200 OK
Set-Cookie: acme_sid=REDACTED; Path=/; Secure; HttpOnly; Expires=Fri, 20 Mar 2026

→ no SameSite attribute.
→ /account/address and /account/preferences accept a cross-origin form post
    with no token; /account/email and /account/password correctly reject it.
```

OWASP: A05 Security Misconfiguration **Fix effort:** Trivial **Verified by hand:** yes

LOW — Internal service hostname disclosed in an error header

ES-0114-06 · `X-Upstream` on 5xx responses from `api.shop.example`

Impact

When an upstream service errors, the edge passes through an `X-Upstream` header naming the internal service and its private DNS suffix. The response body is correctly generic — a bare error and a correlation id, no stack trace — so the leak is limited to your internal naming scheme. It is reconnaissance value only; nothing behind those names is reachable from outside.

The fix

Strip `X-Upstream` at the edge on responses leaving your network. Keep it on the internal hop, where it is useful for tracing.

Evidence

```
GET /api/v1/orders/not-a-valid-id

HTTP/2 502
x-upstream: svc-checkout.internal.acme
x-correlation-id: 0f3c1d9a-...
{"error":"upstream_unavailable","correlation_id":"0f3c1d9a-..."}

→ body is clean; the header is not.
→ svc-checkout.internal.acme does not resolve from outside the network.
```

OWASP: A05 Security Misconfiguration **Fix effort:** Trivial **Verified by hand:** yes

INFO — Staff role names and shipping-rate matrix disclosed unauthenticated

ES-0114-07 · `GET /api/v1/config`

Impact

The unauthenticated bootstrap config returns the full set of staff role identifiers (`admin` , `fulfilment` , `support` , `readonly`), the shipping-rate matrix and the feature flags currently enabled. None of it is customer data and none of it is exploitable on its own, but it hands an attacker your authorization vocabulary — useful when probing for a mass-assignment or privilege-escalation path, which is exactly what we tried next (and it held).

The fix

Trim the public config to what the storefront actually needs to render. Move role identifiers and feature flags behind the authenticated session.

Evidence

```
GET /api/v1/config
```

```
{"roles":["admin","fulfilment","support","readonly"],  
  "flags":{"express_checkout":true,"subscriptions":false},  
  "shipping":{"standard":"4.99","express":"12.99","free_over":"75.00"}}
```

OWASP: A05 Security Misconfiguration **Fix effort:** Trivial **Verified by hand:** yes

INFO — lodash 4.17.20 in the client bundle, no reachable sink

ES-0114-08 · `/assets/vendor.[hash].js`

Impact

CVE-2021-23337, command injection via `_.template` . We mapped every call site in the bundle: `_.template` is not used, and no user-controlled object reaches `_.merge` or `_.set` . It is not exploitable as the application stands today. We are reporting it because a dependency audit will flag it, your customers' security questionnaires will ask about it, and it is one refactor away from mattering.

The fix

Bump to 4.17.21. Add dependency scanning to CI so the next one of these arrives as a pull request rather than as a pentest finding.

Evidence

```
GET /assets/vendor.4f21c8e.js  
→ /** @license lodash 4.17.20 | (c) OpenJS Foundation */
```

- 61 lodash call sites in the bundle; none are `_.template`.
- no user-controlled object reaches `_.merge` or `_.set` on the tested routes.

OWASP: A06 Vulnerable & Outdated Components **Fix effort:** Trivial **Verified by hand:** yes

Controls that held

We test these on every engagement. They are listed whether or not they produced a finding, because “we checked and it was fine” is part of what you are paying for.

Target	Result
Cardholder data at the origin	Held — the PSP iframe is correctly integrated; no PAN reaches your servers or logs, and we could not make the checkout page touch one
SQL / command / template injection	Held — 340 payloads across every reachable parameter; no injection of any class
Admin panel exposure	Held — IP-restricted and MFA-enforced; unreachable from an unlisted address even with valid credentials
Order IDOR on the authenticated API	Held — <code>GET /api/v1/orders/{id}</code> enforces ownership; a second customer's order returns <code>404</code> , not <code>403</code>
Profile mass assignment	Held — only <code>name</code> , <code>phone</code> and <code>address</code> are accepted; <code>role</code> , <code>email</code> and <code>credit</code> are silently ignored
Password change	Held — requires the current password (<code>401</code> without it) and invalidates other sessions
Invoice download path traversal	Held — path normalized and restricted; every <code>..</code> escape returns <code>403</code>
User enumeration on login	Held — valid and invalid addresses return an identical error and an indistinguishable response time
Security headers	Held — CSP, HSTS, <code>nosniff</code> and <code>X-Frame-Options</code> all present and correctly scoped
File upload handling	Held — type and magic-byte checked, stored off-origin under a non-guessable key, served with <code>Content-Disposition: attachment</code>

Bottom line: the access-control layer on the authenticated API is well built, and the payment integration is done properly. The problems are in the places where business rules are enforced — discount stacking, a surname treated as a secret, a role claim trusted for eight hours, a webhook signature not checked. Those are design decisions, not missing patches, which is why a scanner would have returned none of them.

Severity strata

Severity	Meaning	Fix timeline
Critical	No auth needed / full data reachable	Immediate — hours
High	Low-skill exploit, sensitive data	This sprint — 48h
Medium	Needs some context, limited reach	2 weeks
Low	Chained / minimal impact	Backlog
Info	No direct impact, hygiene	Backlog

Re-test

Included in the engagement, run once remediation is reported complete.

Finding	Severity	Status	Verified
ES-0114-01 Discount stacking	High	Closed — duplicate codes rejected, total floored at subtotal	2026-01-31
ES-0114-02 Order tracking walkable	Medium	Closed — signed lookup token in the confirmation email; form retired	2026-01-31
ES-0114-03 Stale role claim	Medium	Closed — role re-checked per request; token lifetime cut to 30m	2026-01-31
ES-0114-04 Unsigned webhook	Medium	Closed — HMAC verified against the raw body, constant-time	2026-01-31

Finding	Severity	Status	Verified
ES-0114-05 Missing SameSite	Low	Closed — SameSite=Lax set; CSRF tokens added to both posts	2026-01-31
ES-0114-06 X- Upstream header	Low	Accepted as risk — client keeps it for tracing; noted and agreed	2026-01-31
ES-0114-07 Config disclosure	Info	Accepted as risk — storefront depends on the flag list; revisit next quarter	2026-01-31
ES-0114-08 lodash 4.17.20	Info	Closed — bumped to 4.17.21, dependency scanning added to CI	2026-01-31

Earthshaker Security — “We find the cracks before they do.” Every finding human-verified, every fix specific.