yof

Security report: Full

Independent check of a web app by yof

shop.example

https://shop.example/

Sample: made-up site

Sample: made-up site. This is a sample report for shop.example, a site that does not exist. Every finding in it is made up.

Report type	Security report: Full
App	shop.example
Issue date	2026-11-07
Report ID	5a3b1e00-0000-4000-8000-000000000000
Version	1

Prepared by yof (yof.io)

Check this report
yof.io/verify/5a3b1e00-0000-4000-8000-000000000000
Fingerprint (SHA-256): fb9ff4c2ff530d7b8b05f4622d7e389ab732e25f51676dc2cd21e1f34878fee5

Confidential: prepared for the owner of shop.example

Document control

Version history

v1 Version 1, issued 2026-11-07

Scope

shop.example: the outside of the app, the code of example-shop/storefront, attack results from the owner's CI.

Requested by

The owner of shop.example, through the yof app (order 00000000).

Distribution

Confidential. Prepared for the owner of shop.example, who may share it with the people who need it, such as auditors, insurers or customers. Anyone holding a copy can check it with the QR code.

Signature

Algorithm	none (sample, not signed) (yof-report/v1 sample)
Key ID	-
Fingerprint (SHA-256)	fb9ff4c2ff530d7b8b05f4622d7e389ab732e25f51676dc2cd21e1f34878fee5
Signature value	Sample report: not signed and not in yof's register.

How to verify

1. Scan the QR code on the cover, or open the address printed under it. The page checks yof's signature against the register.
2. To check a file you were sent, choose it on that page: your browser works out its SHA-256 and compares it, the file is not uploaded.
3. With an Ed25519 signature, anyone can also check it offline with yof's public key, over the report ID, version, issue date and content hash.

yof.io/verify/5a3b1e00-0000-4000-8000-000000000000

What this report is and isn't

This report is a check of shop.example at one point in time, as of 2026-11-07. It covers the repository, the dependencies and the live addresses named in its scope section, and nothing else.

It was made by automated rules and tests. Depending on the scope, yof read the code, checked the packages, tested the live site and, with the owner's consent, ran attack tests. The scope section lists exactly what was done and what was not.

It is evidence to support your security and compliance work. It is not a certification, a conformity assessment or legal advice, and it does not assess governance, policies, processes or other duties that are not technical.

A result with no findings does not mean the app has no vulnerabilities. Automated checks cannot find every flaw. Weaknesses nobody knows about yet (zero-days), flaws in business logic and anything outside the scope may remain, and the app may change after the date shown.

As far as the law allows, yof's liability for this report is limited to the price paid for it. Consumer law where you live may not allow some of these limits; where it does not, they do not apply to you.

Contents

Document control 2
What this report is and isn't 2
Executive summary 4
Attestation 5
Scope and method 6
Findings 8
AI security and accuracy 11
Dependencies and SBOM 12
Appendix A: Accepted risks and why 13
Appendix B: Methodology and tools 14
Appendix C: Glossary 15
Appendix D: Software bill of materials (SBOM) summary 16
Appendix E: How to verify 17
Appendix F: Every check run 18

Executive summary

Overall result

High

High: at least one problem should be fixed now.

- Open findings only, by verdict: confirmed (4), needs validation (0). Listed separately: hardening notes (1), rejected (0).
- Accepted risks (2) are listed in the appendix and are not counted as open.
- 1 problem was fixed since yof first saw it.
- Score over the last 2 checks: 58 then 71 (on what each check could test).

Open findings by urgency

4	0	0	0	2
Fix now	Fix this week	Nice to have	For information	Accepted risks

Top risks, in plain words

F-001 minimist 1.2.5 has a known serious vulnerability (package-lock.json)

The weakness is public, with a published fix. Attackers scan for apps that still run the old version.

Fix now | package-lock.json:812 (example-shop/storefront at 0f1e2d3c4b5a)

F-002 Your Stripe secret key is in your page code (/assets/app.js)

Anyone can use it to take payments, give refunds and read your customers' payment details.

Fix now | https://shop.example/assets/app.js

F-003 Anyone can read a table in your database

A stranger can download the rows in this table, such as your customers' names, emails or orders, without signing in.

Fix now | https://shop.example/

In scope

shop.example: the outside of the app, the code of example-shop/storefront, attack results from the owner's CI.
Coverage: 45% of the six parts yof can examine.

Out of scope

- Pages behind a login were not tested: yof signs in only through the owner's own attack runs.
- Only the first pages were read; the rest of the site was not.
- AI assistant: not tested in this report.

Attestation

This letter records that yof ran the checks listed in this report against shop.example (https://shop.example/) between 2026-11-07 and 2026-11-07, for report 5a3b1e00-0000-4000-8000-000000000000, version 1, issued 2026-11-07.

Scope: the outside of the app, the code of example-shop/storefront, attack results from the owner's CI. The owner proved control of the app before any deep check ran. Coverage: 45% of the six parts yof can examine (outside, inside the code, pages behind a login, attack tests, the AI assistant, every page).

Method: each check follows the published method named next to it in section 3 (OWASP WSTG v4.2, OWASP ASVS 4.0.3, the OWASP Top 10 for LLM Applications 2025, and the DNS standards for email). Every request yof made is in the proof list in the appendix.

Limits: findings describe the app as it was on those dates. Parts that were not tested are listed, never assumed. This letter is evidence of testing; it is not an audit opinion and makes no statement about anything outside its scope.

Integrity: the SHA-256 hash of this report's content and its signature are on the last page. Anyone can check the report ID and hash against yof's public register; a changed copy will not match.

Scope and method

Coverage: 45% of the six parts yof can examine.

- Outside: Tested in part
- Inside the code: Tested in full
- Pages behind a login: Not tested
- Attack tests: Tested in full
- AI assistant: Not tested
- Every page: Not tested

Checks and their methods

- Keys in page code: Ran in full, things checked: 14. Method: WSTG-INFO-05, ASVS V14.3. Read the code every visitor downloads and look for keys and passwords in it.
- Database access: Ran in full, things checked: 4. Method: WSTG-ATHZ-02, ASVS V4.1. Ask the app's own database service, without signing in, for rows a visitor should not see.
- Files that should be private: Ran in full, things checked: 42. Method: WSTG-CONF-04, WSTG-CONF-03. Ask for files that should never be public: backups, environment files, source maps, repository folders.
- Admin pages: Ran in full, things checked: 9. Method: WSTG-CONF-05. Look for admin and debug pages that answer without a sign-in.
- Public API: Ran in part, things checked: 6. Method: WSTG-CONF-06, WSTG-ATHZ-02, OWASP API Top 10 2023. Call the app's public API addresses as a stranger and read what they return.
- Browser protection headers: Ran in full, things checked: 1. Method: WSTG-CONF-07, WSTG-CONF-12, WSTG-CLNT-09. Read the response headers for HTTPS enforcement, framing, content type and referrer rules.
- Email sending rules: Ran in full, things checked: 2. Method: RFC 7208 (SPF), RFC 6376 (DKIM), RFC 7489 (DMARC). Read the domain's DNS records for the rules that stop others sending mail as it.
- Website in a real browser: Ran in full, things checked: 1. Website basics. Not a security test.
- Packages: Ran in full, things checked: 640. Method: ASVS V14.2.1, OWASP A06:2021. Read every lockfile and ask OSV.dev about each package version.
- Keys in the code: Ran in full, things checked: 214. Method: ASVS V2.10.4, ASVS V6.4.1. Read every text file for keys, tokens and passwords.
- Configuration: Ran in full, things checked: 214. Method: ASVS V14.1, ASVS V14.4, WSTG-CONF-12. Read Dockerfiles, CI workflows, hosting config, database migrations and access rules.
- Code: Ran in full, things checked: 170. Method: ASVS V5, WSTG-INPV. Read the code for injection, open redirects, unchecked tokens and weak password storage.
- Attack tests: Ran in full, things checked: 1. Method: WSTG-ATHZ-01, WSTG-ATHZ-02, WSTG-ATHZ-04, WSTG-ATHN-04, WSTG-SESS-01. Results the owner's own CI uploaded from yof attack: signed-in requests as other users and roles. Uploaded 2026-11-06.

Not tested

- Pages behind a login were not tested: yof signs in only through the owner's own attack runs.
- Only the first pages were read; the rest of the site was not.
- AI assistant: not tested in this report.

Standards referenced

ASVS V14.1, ASVS V14.2.1, ASVS V14.3, ASVS V14.4, ASVS V2.10.4, ASVS V4.1, ASVS V5, ASVS V6.4.1, OWASP A06:2021, OWASP API Top 10 2023, RFC 6376 (DKIM), RFC 7208 (SPF), RFC 7489 (DMARC), WSTG-ATHN-04, WSTG-ATHZ-01, WSTG-ATHZ-02, WSTG-ATHZ-04, WSTG-CLNT-09, WSTG-CONF-03, WSTG-CONF-04, WSTG-CONF-05, WSTG-CONF-06, WSTG-CONF-07, WSTG-CONF-12, WSTG-INFO-05, WSTG-INPV, WSTG-SESS-01

What was checked and what was not

This states the reach of this report. A part that was not checked is not assumed to be fine.

Checked

- Outside: Tested in part
- Inside the code: Tested in full
- Attack tests: Tested in full

Not checked, and why

- Pages behind a login were not tested: yof signs in only through the owner's own attack runs.
- Only the first pages were read; the rest of the site was not.
- AI assistant: not tested in this report.

Never covered by automated checks

- Flaws in business logic, such as a price or a step a person can skip.
- Weaknesses nobody knows about yet (zero-days).
- People and processes: phishing, passwords shared by hand, who has access.
- Anything outside the scope named in this report, and any change after its date.

How sure yof is of each finding

Every finding carries one verdict. Only confirmed findings and findings that need validation are counted. Hardening notes and rejected findings are listed apart and never counted.

- Confirmed: yof saw the impact happen, or holds the exact evidence (for example the key that was served, or the installed version an advisory names).
- Needs validation: Likely, but not demonstrated: yof saw the pattern and could not prove the impact. A person or a deeper test needs to confirm it.
- Rejected: Disproved by a second, separate check. Kept for the record, not counted.

What each severity means

- Critical: Only for a confirmed finding that yof saw happen: data or control that belongs to someone else (another person's records, a whole table, a key with full power served to everyone), or a package known to be exploited or malicious.
- High: Serious harm is likely if someone uses it: a confirmed exposure with limits, or a likely problem yof could not demonstrate. A finding that is not confirmed can reach high at most.
- Medium: Makes an attack easier or its damage larger, under conditions such as a sign-in, a user's click or a specific setting.
- Low: Small harm on its own, or reachable only in development and test tools.
- Info: Worth knowing, no harm on its own.

Findings

4 open: 4 fix now, 0 fix this week, 0 nice to have, 0 for information.
Open findings only, by verdict: confirmed (4), needs validation (0). Listed separately: hardening notes (1), rejected (0).

Fix now | CVSS 9.3 Critical | Open

F-001 minimist 1.2.5 has a known serious vulnerability (package-lock.json)

Verdict: Confirmed. Severity: High. Why: The repository holds the exact evidence. yof did not see it on the live app, so it is rated high at most.

The weakness is public, with a published fix. Attackers scan for apps that still run the old version.

CWE-1395 | OWASP A06:2021 Vulnerable and Outdated Components

CVSS v4.0 vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N (typical for this kind of finding; the advisory has the exact score)

Where: package-lock.json:812 (example-shop/storefront at 0f1e2d3c4b5a)

Steps to see it:

1. Open package-lock.json:812 (example-shop/storefront at 0f1e2d3c4b5a).
2. Expected instead: The installed version of minimist is at or above 1.2.6.

Business impact: A published weakness in a component could be used against the app.

Package: minimist (npm)

Installed version: 1.2.5

Advisories: GHSA-xvch-5gv4-984h https://github.com/advisories/GHSA-xvch-5gv4-984h, CVE-2021-44906 https://nvd.nist.gov/vuln/detail/CVE-2021-44906

First fixed version: 1.2.6

Lockfile: package-lock.json:812

Named in: package.json

Fix: Upgrade minimist from 1.2.5 to 1.2.6 or newer (advisories: GHSA-xvch-5gv4-984h, CVE-2021-44906). Update package-lock.json with your package manager rather than by hand, run the install and the tests, and check whether the upgrade needs code changes.

References:

- GHSA-xvch-5gv4-984h (https://github.com/advisories/GHSA-xvch-5gv4-984h)
- CVE-2021-44906 (https://nvd.nist.gov/vuln/detail/CVE-2021-44906)
- CWE-1395: Dependency on Vulnerable Third-Party Component (https://cwe.mitre.org/data/definitions/1395.html)
- OWASP A06:2021: Vulnerable and Outdated Components

First seen 2026-10-30. Last seen 2026-11-07.

Fix now | CVSS 9.3 Critical | Open

F-002 Your Stripe secret key is in your page code (/assets/app.js)

Verdict: Confirmed. Severity: Critical. Why: yof saw it on the live app.

Anyone can use it to take payments, give refunds and read your customers' payment details.

CWE-798 | OWASP A07:2021 Identification and Authentication Failures

CVSS v4.0 vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:H/SI:H/SA:H

Where: https://shop.example/assets/app.js

Evidence (masked): sk_live_****9f2a

Steps to see it:

1. Open https://shop.example/assets/app.js without signing in.
2. What yof saw (masked): sk_live_****9f2a
3. Expected instead: No live Stripe secret key appears in code that visitors can download.

Business impact: Someone could sign in as another person or keep a session they should not have.

Fix: Remove the Stripe secret key (it starts with sk_live_) from all browser code and from any environment variable the browser can read. Create payments on the server only (an API route, server action or Supabase Edge

Function) and send the browser just the client secret of the payment. The browser should only ever see the publishable key (pk_live_). Never paste the real key into this chat, never print it, and do not invent a replacement key. Tell me when the code no longer contains it, so I can create a new key in the provider's dashboard and delete the old one (the old one has been public, so it must be replaced).

References:

CWE-798: Use of Hard-coded Credentials (https://cwe.mitre.org/data/definitions/798.html)
OWASP A07:2021: Identification and Authentication Failures

First seen 2026-10-30. Last seen 2026-11-07.

Fix now | CVSS 8.7 High | Open

F-003 Anyone can read a table in your database

Verdict: Confirmed. Severity: High. Why: yof saw it on the live app.

A stranger can download the rows in this table, such as your customers' names, emails or orders, without signing in.

CWE-862 | OWASP A01:2021 Broken Access Control
CVSS v4.0 vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

Where: https://shop.example/

Evidence (masked): table orders: 3 rows readable without signing in

Steps to see it:

1. Open https://shop.example/ without signing in.
2. What yof saw (masked): table orders: 3 rows readable without signing in
3. Expected instead: Visitors who are not signed in cannot read this table.

Business impact: People could see or change data that belongs to someone else.

Fix: A Supabase table can be read by anyone with the public anon key. Turn on Row Level Security for this table (and every other table), then add policies so a signed-in person can only read their own rows (for example using auth.uid() = user_id). Remove any policy that allows the anon role to select everything. Write the SQL for me and explain what each policy allows. Never paste real customer data or keys into this chat.

```
-- for each table the browser can reach
alter table public.your_table enable row level security;
create policy "own rows" on public.your_table
  for select using (auth.uid() = user_id);
```

References:

CWE-862: Missing Authorization (https://cwe.mitre.org/data/definitions/862.html)
OWASP A01:2021: Broken Access Control

First seen 2026-10-30. Last seen 2026-11-07.

Fix now | CVSS 7.1 High | Open

F-004 One user can read another user's records by changing the id

Verdict: Confirmed. Severity: High. Why: yof saw it on the live app.

A test account opened a record of your other test account just by using its id, so real users can read each other's data.

CWE-639 | OWASP A01:2021 Broken Access Control | OWASP API1:2023 Broken Object Level Authorization
CVSS v4.0 vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

Where: GET /api/orders/{id} (as user b)

Evidence (masked): 200 with another customer's order

Steps to see it:

1. Sign in as test user b.
2. Send GET /api/orders/{id} with an id that belongs to another user or role.
3. What yof saw (masked): 200 with another customer's order
4. Expected instead: An account can open only its own records; another account's id answers 404 or 403.

Business impact: People could see or change data that belongs to someone else.

Fix: In my app, user B can read a record that belongs to user A by putting A's record id in the address (or sees it in a list). On the server, every query for a record or a list must be filtered by the signed-in user (or their workspace),

for example `where id = :id and owner_id = :currentUser`, and answer 404 when the record is not theirs. If the database supports row-level security, turn it on with a policy that only allows the owner. Do not rely on ids being hard to guess.

References:

CWE-639: Authorization Bypass Through User-Controlled Key (https://cwe.mitre.org/data/definitions/639.html)
OWASP A01:2021: Broken Access Control
OWASP API1:2023: Broken Object Level Authorization

First seen 2026-10-30. Last seen 2026-11-07.

Fix this week | CVSS 5.1 Medium | Fixed

F-005 Your domain has no rule for fake email

Verdict: Needs validation. Severity: Medium. Why: The check did not keep the evidence needed to confirm it.
Mail providers don't know what to do with messages that pretend to be you, so many of them get delivered.
CWE-290 | OWASP A05:2021 Security Misconfiguration
CVSS v4.0 vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:A/VC:N/VI:N/VA:N/SC:L/SI:L/SA:N
Where: https://shop.example/

Steps to see it:

1. Look up the domain's DNS TXT records (for example: dig TXT shop.example).
2. Expected instead: The domain publishes a DMARC policy of quarantine or reject.

Business impact: A setting leaves an opening that is easy to find.
Fix: The domain has no DMARC record. Add a TXT record whose name (host) is exactly _dmarc, not @ (a DMARC record at the root is ignored), starting with monitoring: v=DMARC1; p=none; rua=mailto:<an address you read>. After a few weeks of clean reports, move to p=quarantine and then p=reject.

```
_dmarc.shop.example.  TXT  "v=DMARC1; p=quarantine; rua=mailto:dmarc@shop.example"
```

References:

CWE-290: Authentication Bypass by Spoofing (https://cwe.mitre.org/data/definitions/290.html)
OWASP A05:2021: Security Misconfiguration

First seen 2026-10-30. Last seen 2026-11-07. Fixed 2026-11-04.

Hardening notes (not counted as findings)

Good practice that would add a layer of protection. yof demonstrated no risk, so these are not findings and are in no count.

- H-001 Browsers aren't told to always use https: https://shop.example/

  Add the Strict-Transport-Security header to every response, for example: Strict-Transport-Security: max-age=31536000; includeSubDomains. Set it in the host or framework headers config.

```
// vercel.json (or the same header in your host's settings)
{
  "headers": [
    { "source": "/(.*)", "headers": [ { "key": "Strict-Transport-Security", "value": "max-a
ge=63072000; includeSubDomains" } ] }
  ]
}
```

- H-002 No rule limits which scripts can run: https://shop.example/

  Add a Content-Security-Policy header that lists where scripts, styles, images and connections may come from. Start in Report-Only mode, check that the app still works, then enforce it.

```
// vercel.json (or the same header in your host's settings)
{
  "headers": [
    { "source": "/(.*)", "headers": [ { "key": "Content-Security-Policy", "value": "default
-src 'self'; frame-ancestors 'self'; object-src 'none'; base-uri 'self'" } ] }
  ]
}
```

AI security and accuracy

What yof tested in the AI parts of this app, by the OWASP Top 10 for LLM Applications 2025, and how often the assistant's answers matched the owner's own pages. Probes are fixed and mild, sent only with the owner's consent; answers are read by fixed rules, never by another model.

What was tested

- The assistant, with the owner's consent: no.
- Attack results from the owner's own runner: yes.
- AI coding tools and packages in the repository: yes.

OWASP Top 10 for LLM Applications 2025

- LLM01 Prompt Injection: Tested: nothing open
- LLM02 Sensitive Information Disclosure: Tested: nothing open
- LLM03 Supply Chain: Tested: nothing open
- LLM04 Data and Model Poisoning: Tested: nothing open
- LLM05 Improper Output Handling: Tested: nothing open
- LLM06 Excessive Agency: Tested: nothing open
- LLM07 System Prompt Leakage: Tested: nothing open
- LLM08 Vector and Embedding Weaknesses: Not tested by yof
- LLM09 Misinformation: Not tested by yof
- LLM10 Unbounded Consumption: Tested: nothing open

Not tested

- The assistant was not tested in this run: no setup with the owner's consent.
- Vector store and embedding weaknesses (OWASP LLM08) are not tested by yof.

Dependencies and SBOM

Repository: example-shop/storefront at 0f1e2d3c4b5a
Components: 640
With a known advisory: 1
- npm: 640
- Lockfile: package-lock.json

SHA-256 of the sorted package URLs:
5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a

- F-001 minimist 1.2.5: GHSA-xvch-5gv4-984h, CVE-2021-44906, first fixed version 1.2.6 (package-lock.json:812)

The full list of components is in the CycloneDX 1.5 file that comes with this report (SBOM).

Appendix A: Accepted risks and why

The owner looked at these findings and decided to accept the risk, for the reason given. They are not counted as open, and they are listed here so nothing is hidden.

H-002 No rule limits which scripts can run

Where: https://shop.example/
Reason: The payment widget needs inline scripts until its next version; we review this in January.
Accepted by: owner@shop.example, 2026-11-05
Accepted in: the yof app

A Stripe test secret key is in your page code

Where: tests/fixtures/payments.ts
Reason: A Stripe test key in a test fixture; it cannot take real payments.
Accepted by: Sam (developer), 2026-11-02
Accepted in: the repository (.yof/accepted.json)

Appendix B: Methodology and tools

yof ran its own rules and tests: the outside check of the live app, a read of the linked repository and its packages, the owner's own attack runs, and, with consent, fixed probes of the AI assistant. Each check follows the published method named next to it.

No outside scanner or AI model judged the findings. Evidence is masked before it is stored, and every request yof made is in the proof list.

- Keys in page code: Ran in full (WSTG-INFO-05, ASVS V14.3)
- Database access: Ran in full (WSTG-ATHZ-02, ASVS V4.1)
- Files that should be private: Ran in full (WSTG-CONF-04, WSTG-CONF-03)
- Admin pages: Ran in full (WSTG-CONF-05)
- Public API: Ran in part (WSTG-CONF-06, WSTG-ATHZ-02, OWASP API Top 10 2023)
- Browser protection headers: Ran in full (WSTG-CONF-07, WSTG-CONF-12, WSTG-CLNT-09)
- Email sending rules: Ran in full (RFC 7208 (SPF), RFC 6376 (DKIM), RFC 7489 (DMARC))
- Website in a real browser: Ran in full
- Packages: Ran in full (ASVS V14.2.1, OWASP A06:2021)
- Keys in the code: Ran in full (ASVS V2.10.4, ASVS V6.4.1)
- Configuration: Ran in full (ASVS V14.1, ASVS V14.4, WSTG-CONF-12)
- Code: Ran in full (ASVS V5, WSTG-INPV)
- Attack tests: Ran in full (WSTG-ATHZ-01, WSTG-ATHZ-02, WSTG-ATHZ-04, WSTG-ATHN-04, WSTG-SESS-01)

Methods and standards referenced

ASVS V14.1, ASVS V14.2.1, ASVS V14.3, ASVS V14.4, ASVS V2.10.4, ASVS V4.1, ASVS V5, ASVS V6.4.1, OWASP A06:2021, OWASP API Top 10 2023, RFC 6376 (DKIM), RFC 7208 (SPF), RFC 7489 (DMARC), WSTG-ATHN-04, WSTG-ATHZ-01, WSTG-ATHZ-02, WSTG-ATHZ-04, WSTG-CLNT-09, WSTG-CONF-03, WSTG-CONF-04, WSTG-CONF-05, WSTG-CONF-06, WSTG-CONF-07, WSTG-CONF-12, WSTG-INFO-05, WSTG-INPV, WSTG-SESS-01

Appendix C: Glossary

Finding
Something yof saw that could let an attacker in or expose data, with where it is and how to see it.

Urgency
yof's plain order of work: fix now, fix this week, nice to have, for information.

CVSS 4.0
The common scoring system for how serious a weakness is, from 0 to 10, with the vector that explains the score.

CWE
The common list of weakness types; each finding names the type it belongs to.

Coverage
How much of the six parts yof can examine were tested in this report.

Accepted risk
A finding the owner decided to keep, with a reason. It is listed, not counted as open.

SBOM
Software bill of materials: the list of packages the app is built from.

Fingerprint
The SHA-256 hash of the report's content. Any change to the report changes it.

Appendix D: Software bill of materials (SBOM) summary

Packages in the SBOM file that comes with this report: 640. With a known advisory: 1.

- example-shop/storefront @ 0f1e2d3c4b5a
- npm: 640
- package-lock.json

SHA-256 5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a

Appendix E: How to verify

1. Scan the QR code on the cover, or open the address printed under it. The page checks yof's signature against the register.
2. To check a file you were sent, choose it on that page: your browser works out its SHA-256 and compares it, the file is not uploaded.
3. With an Ed25519 signature, anyone can also check it offline with yof's public key, over the report ID, version, issue date and content hash.

- Files: pdf, json, sarif, findings_json, xlsx, jira_csv, linear_csv, cyclonedx.
- Report ID: 5a3b1e00-0000-4000-8000-000000000000. Version: 1. Issued: 2026-11-07T09:00:00.000Z.
- The JSON file holds this whole report; its content hash is the SHA-256 of the canonical JSON without the signature block.
- Check a copy against yof's public register: GET https://api.mondaily.com/api/v1/public/yof-reports/verify?id=<report id>&sha=<hash>. The hash of the PDF or any other file of this report is accepted too.
- Or scan the QR code on the cover, or open https://yof.io/verify/5a3b1e00-0000-4000-8000-000000000000: the page checks the signature and shows only what the report's owner chose to share.

Check this report
yof.io/verify/5a3b1e00-0000-4000-8000-000000000000
Fingerprint (SHA-256): fb9ff4c2ff530d7b8b05f4622d7e389ab732e25f51676dc2cd21e1f34878fee5

Limits

- A report shows evidence of what was tested on the dates above. It is not a promise about anything that was not tested, or about later changes.
- Evidence is masked: keys, tokens, passwords and email addresses are never printed in full.
- Coverage was 45%. Parts that were not tested are listed in section 3.

Appendix F: Every check run

- Keys in page code: Ran in full, things checked: 14
- Database access: Ran in full, things checked: 4
- Files that should be private: Ran in full, things checked: 42
- Admin pages: Ran in full, things checked: 9
- Public API: Ran in part, things checked: 6
- Browser protection headers: Ran in full, things checked: 1
- Email sending rules: Ran in full, things checked: 2
- Website in a real browser: Ran in full, things checked: 1
- Packages: Ran in full, things checked: 640
- Keys in the code: Ran in full, things checked: 214
- Configuration: Ran in full, things checked: 214
- Code: Ran in full, things checked: 170
- Attack tests: Ran in full, things checked: 1

Proof list

One line for each thing yof looked at in the outside run, and the result (lines: 4).

bundle | script | https://shop.example/assets/app.js | found something
headers | header | https://shop.example/ | ok
files | address | https://shop.example/.env (x42) | ok
database | table | orders | found something