

This is a fictional example. It is not a real client or real findings.

SAMPLE REPORT

What a Confidence Report looks like.

EXAMPLE REPORT, FICTIONAL

EXAMPLE, FICTIONAL

Confidence Report

Example Platform, Example Co

Client	Example Co	Product	Example Platform
Date	Illustrative	Version reviewed	Illustrative
Prepared by	Nash Software Services		

Executive summary

EXAMPLE, FICTIONAL

CONFIDENCE RATING

Conditional

VERDICT

Fix first, then continue.

In plain English

1. Example Platform is worth keeping. The structure is sound and most of the code is in reasonable shape for a product of its age.
2. Three problems need fixing before more customers are added: one customer can see another customer's records, live passwords are stored in the code, and payment updates can be faked or applied twice.
3. Fix those three first, in the order below, then continue building. Nothing found points to a rebuild.

Status by area

AMBER

Ownership and access

The hosting account sits under the previous agency's login. Move it to a company account.

RED

Security basics

Customer records are not checked against the signed-in account, and live secrets are in the code.

AMBER

Code health

Readable and consistent, with some duplicated logic in the billing area.

RED

Data handling

Payment updates are trusted without checking they came from the payment provider.

AMBER

Testing and releases

Tests exist and pass, but they skip permissions and payments. Releases are manual.

GREEN

Scalability

Comfortable at current and near-term load. Two slow queries to watch as data grows.

Findings at a glance

EXAMPLE, FICTIONAL

Material risk

3

Fix before adding more customers.
Ranked by severity.

Fine for now

6

Acceptable at the current size.
Revisit as you grow.

Two reporting queries slow down as the orders table grows. Add indexes before it reaches a size where users notice.

Error logs are written but nobody is alerted. Set up alerts before the team relies on them.

Preference

9

A matter of style or approach. Your developer's call.

Folder naming mixes two conventions. Pick one when files are next touched.

Some date handling uses a helper and some does not. Harmless, but one way would be easier to read.

Material risks, by severity

EXAMPLE, FICTIONAL

An extract from the detailed findings. The full report has every finding in all three groups.

MR-1 · RANK 1 **CRITICAL** SECURITY BASICS

Customers can read other customers' records

What I found

The API checks that someone is signed in, but not that the record they ask for belongs to their account. Changing the record number in a request to the invoices and projects endpoints returns another customer's data.

Why it matters

Any customer can see any other customer's invoices and project details. That is a data breach waiting for someone curious enough to try it, with the reporting duties and loss of trust that follow.

Evidence

File and line references, with the request used to reproduce it, are in the full report.

Recommended fix

1. Add one shared check that every record lookup is limited to the signed-in account.
2. Apply it to every endpoint that takes a record number, starting with invoices and projects.
3. Add tests that try to read another account's records and expect a refusal.

Rough effort

2 to 4 days (illustrative)

Live secrets are stored in the code

What I found

The production database password and the payment provider's live key are committed in a configuration file, and the same values are used in staging and production.

Why it matters

Anyone with access to the code, now or in the past, can reach the live database and the live payment account. Staging is less protected, so a problem there becomes a problem in production.

Evidence

File and line references, with the request used to reproduce it, are in the full report.

Recommended fix

1. Replace the database password and the payment key now, and treat the old values as exposed.
2. Move secrets into the hosting platform's secret settings, with separate values for staging and production.
3. Add a check to the release process that rejects commits containing secrets.

Rough effort

1 to 2 days (illustrative)

Payment updates are not verified and can apply twice

What I found

The endpoint that receives payment notifications does not check the provider's signature, so anyone who finds the address can mark an order as paid. It also does not record which notifications it has already handled, so a repeated notification is processed again.

Why it matters

Orders can be marked as paid without payment, and a retried notification can send a customer two confirmations or credit their account twice. Both are hard to untangle once they reach your accounts.

Evidence

File and line references, with the request used to reproduce it, are in the full report.

Recommended fix

1. Verify the provider's signature on every notification and reject anything that fails.
2. Store each notification's identifier and skip any that has already been handled.
3. Add tests for a forged notification and a repeated one.

Rough effort

2 to 3 days (illustrative)

What I looked at and how

EXAMPLE, FICTIONAL

- Material supplied: read-only access to the code repository, a read-only cloud role, a staging environment with seeded data, and the team's architecture notes. I changed nothing.
- The product, as described in this example: a web application written in TypeScript, with a Node.js API, a PostgreSQL database and a hosted payment provider.

- I read the code, ran the application and its tests myself against staging, and checked the hosting and release configuration.
- I formed my own view from the code, the running system and the configuration before discussing design with the developer. Questions to the developer before that were factual setup questions only.
- Findings are sorted into material risk, fine for now and preference. Material risks are ranked by severity. Each finding has evidence and a recommended fix your developer can act on.

What this report is not

EXAMPLE, FICTIONAL

It is not legal, regulatory or investment advice.

It is based on the material supplied and the version reviewed. Anything not supplied was not reviewed.

It is a point-in-time view. Code and configuration change, and the findings describe the system as it was when reviewed.

It is not a penetration test of the live system. The review reads and runs the code; it does not attack production.