

 RESOURCE

Sample penetration test report

This is what you receive after a web application and API penetration test: an executive summary, the scope, the method, each finding with evidence and a fix, and the retest results. Every name, system, finding and date in it is invented for illustration.

AT A GLANCE

Report

Web application and API penetration test

Organization

Example Care Co. (invented)

Approach

Gray-box, three test accounts

Status

Retest complete



This report is synthetic

The organization, systems, requests, findings and dates are all invented. Hostnames use the reserved name `example.test`, which cannot point to a real system. It does not describe a real client or real results. A real report depends on the engagement: its scope, the systems and what we find.

Executive summary

Example Care Co. asked for a gray-box penetration test of its customer portal and the API behind it before a major release. Testing ran over ten business days in a staging environment, using three test accounts: a standard user in each of two test organizations and an administrator in one.

We found six issues: two high, two medium, one low and one informational. None allowed access without signing in. The two high-risk issues matter most. Any signed-in user could read another organization's invoices by changing a number in a request, and a standard user could make themselves an administrator by adding one field to a profile update.

The cause is the same in both: the API trusts what the request says instead of checking what the caller is allowed to do. Both have clear fixes in application code. At the retest, four of the six issues were fixed and verified, one was partly fixed and one low-priority item is scheduled for the next release.

Key points

- ✓ Two high, two medium, one low, one informational
- ✓ No access without signing in
- ✓ Both high issues are authorization flaws in the API
- ✓ Fix the shared authorization check first
- ✓ Four of six fixed and verified at retest

Scope and rules of engagement

Item	Detail
In scope	Customer web portal at portal.example.test. Public REST API at api.example.test, with its OpenAPI specification.
Out of scope	The production environment. The third-party payment processor. Denial-of-service testing. Social engineering.
Environment	A staging copy of the application with synthetic data.
Accounts provided	Standard user in organization A. Standard user in organization B. Administrator in organization A.
Approach	Gray-box: the testers had accounts and the API specification, but no source code.
Window	Ten business days, weekdays 09:00 to 18:00 Eastern Time.
Authorization	Signed scope and rules of engagement on file before testing started. Named emergency contacts on both sides.

How we tested

STEP 1

Plan and scope

Agree the scope, accounts, rules and test window in writing.

STEP 2

Map the application

Walk every feature as each role. Build a list of endpoints and the data each one returns.

STEP 3

Test by category

Authentication, authorization, input handling, session management and business logic, using manual techniques and tooling.

STEP 4

Validate and chain

Confirm every finding by hand and try combining issues to see how far an attacker could get.

STEP 5

Report and retest

Report with evidence and fixes. Retest after the client fixes to confirm.

Testing followed the OWASP Web Security Testing Guide and the OWASP API Security Top 10 (2023), with NIST SP 800-115 as the general method. Severity uses CVSS v3.1.

Findings

Choose a severity to filter. Open a finding to see the evidence, why it matters and how to fix it.

0
CRITICAL

2
HIGH

2
MEDIUM

1
LOW

1
INFORMATIONAL

ID	FINDING	SEVERITY	CVSS	RETEST
ECC-01	Any signed-in user can read other organizations' invoices	HIGH	8.1	Fixed and verified
ECC-02	A standard user can make themselves an administrator	HIGH	8.8	Fixed and verified
ECC-03	The password reset form reveals which email addresses have	MEDIUM	5.3	Fixed and verified

ID	FINDING	SEVERITY	CVSS	RETEST
	accounts			
ECC-04	Sessions stay valid after a password change	MEDIUM	5.4	Partly fixed
ECC-05	Error messages reveal internal details	LOW	3.7	Fixed and verified
ECC-06	Recommended browser security headers are missing	INFORMATIONAL		Still open

HIGH**ECC-01 Any signed-in user can read other organizations' invoices****AFFECTED** GET /v2/invoices/{id} (api.example.test)**CVSS V3.1** 8.1 AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N**RETEST** Fixed and verified. The server now rejects requests for another organization's invoice with a 404. Verified with both test accounts.**What we found**

The invoice endpoint checks that the caller is signed in but not that the invoice belongs to the caller's organization. Invoice numbers are sequential, so a signed-in user of one organization can read every other organization's invoices by counting upward. This is a broken object-level authorization flaw.

Evidence**SYNTHETIC**

Request, signed in as a user of organization A:

GET /v2/invoices/48213 HTTP/1.1

Host: api.example.test

Authorization: Bearer <token for user-a@org-a.example.test>

Response:

HTTP/1.1 200 OK

{"id":48213,"organization":"org-b","total":"1,240.00","customer":"[redacted]"}

Invoice 48213 belongs to organization B.

Why it matters

Invoices contain customer names, addresses and amounts. One customer's data is exposed to every other customer. The test used only the two test organizations, and the same weakness would apply to every real customer.

How likely: High. Any customer with an account can do it, with no special tools, by changing a number in a request.

How to fix it

1. On every request, compare the invoice's organization with the caller's organization on the server, and return 404 when they differ.
2. Put that check in shared code that every object lookup passes through, so new endpoints inherit it.
3. Use identifiers that cannot be guessed, as defense in depth. Do not rely on them instead of the check.
4. Add an automated test that signs in as organization A and requests organization B's objects on every endpoint.

References

- OWASP API Security Top 10 (2023): API1 Broken Object Level Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key

HIGH**ECC-02 A standard user can make themselves an administrator****AFFECTED** PATCH /v2/users/me (api.example.test)**CVSS V3.1** 8.8 AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H**RETEST** Fixed and verified. The endpoint now accepts only an allow-list of profile fields. Role changes require an administrator and are logged.**What we found**

The profile update endpoint copies every field in the request into the user record. Adding a role field to an ordinary update changes the caller's role. This is a broken object property level authorization flaw, often called mass assignment.

Evidence**SYNTHETIC**

```

PATCH /v2/users/me HTTP/1.1
Host: api.example.test
Authorization: Bearer <token for a standard user>
Content-Type: application/json

{"display_name":"Test","role":"org_admin"}

HTTP/1.1 200 OK
{"id":1042,"display_name":"Test","role":"org_admin"}

```

Why it matters

A standard user becomes an administrator of their own organization. They can then manage users and billing settings. Combined with ECC-01, it widens what an attacker inside one customer account can do.

How likely: High. It needs one added field in a request that the application already sends.

How to fix it

1. Accept only an explicit allow-list of fields on each update endpoint. Reject unknown fields.

2. Keep sensitive properties such as role, organization and status out of user-editable models.
3. Require an administrator for role changes and write each change to the audit log.

References

- OWASP API Security Top 10 (2023): API3 Broken Object Property Level Authorization
- CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

MEDIUM ECC-03

The password reset form reveals which email addresses have accounts

AFFECTED POST /auth/forgot-password (portal.example.test)

CVSS V3.1 5.3 AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

RETEST Fixed and verified. Both cases now show the same message and take the same time. Requests are rate limited.

What we found

The reset form answers “no account found” for unknown addresses and “email sent” for known ones, and there is no limit on how many addresses can be tried. An attacker can build a list of valid customer accounts.

Evidence SYNTHETIC

```
POST /auth/forgot-password email=nobody@example.test
```

```
→ 404 {"error":"No account found"}
```

```
POST /auth/forgot-password email=user-a@org-a.example.test
```

```
→ 200 {"message":"Reset email sent"}
```

```
200 requests in 60 seconds returned no rate-limit response.
```

Why it matters

A list of valid accounts makes phishing and password guessing more targeted. It exposes which companies are customers.

How likely: High that it can be done. Moderate impact on its own.

How to fix it

1. Return one message for every address, such as “If an account exists, we sent an email”, and keep the response time the same.
2. Rate limit reset requests by address and by source.
3. Enable multi-factor authentication so a known address is less useful on its own.

References

- OWASP WSTG: Testing for Account Enumeration and Guessable User Account (WSTG-IDNT-04)
- CWE-204: Observable Response Discrepancy

MEDIUM**ECC-04 Sessions stay valid after a password change****AFFECTED** Session handling (portal.example.test)**CVSS V3.1** 5.4 AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N**RETEST** Partly fixed. Web sessions now end on password change. Mobile API tokens still live for 30 days. Fix scheduled.**What we found**

After a user changes their password, earlier session tokens keep working. Someone who stole a session still has access after the owner reacts.

Evidence**SYNTHETIC**

1. Sign in on browser 1 and copy the session cookie.
2. Sign in on browser 2 and change the password.
3. Replay the old cookie from browser 1:

```
GET /account HTTP/1.1  
Cookie: session=<old value>
```

```
HTTP/1.1 200 OK
```

Why it matters

A stolen session survives the owner's response. It limits the value of resetting a password after a suspected compromise.

How likely: Moderate. It needs a stolen token first.

How to fix it

1. Invalidate all other sessions and refresh tokens when a password changes.
2. Shorten the lifetime of long-lived tokens and bind them to the device where practical.
3. Show users their active sessions and let them end any of them.

References

- CWE-613: Insufficient Session Expiration
- OWASP Cheat Sheet: Session Management

LOW**ECC-05 Error messages reveal internal details****AFFECTED** POST /v2/reports (api.example.test)**CVSS V3.1** 3.7 AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N**RETEST** Fixed and verified. Errors now return a generic message and a correlation ID. Details stay in server logs.**What we found**

A malformed request to the reports endpoint returns a stack trace with internal class names, a framework version and an internal hostname.

Evidence**SYNTHETIC**

```
POST /v2/reports HTTP/1.1
{"range": {"from": "not-a-date"}}

HTTP/1.1 500 Internal Server Error
{"error": "DateParseException at ReportService.build (reports-svc-03.internal.example.test) ..."}

```

Why it matters

The detail helps an attacker choose techniques and map internal systems. It is not dangerous alone.

How likely: Low to moderate. It helps only as part of a larger attack.

How to fix it

1. Return a generic error and a correlation ID to the caller. Log the detail on the server.
2. Turn off debug output in production and add a test that fails if a stack trace appears in a response.

References

- CWE-209: Generation of Error Message Containing Sensitive Information
- OWASP WSTG: Testing for Improper Error Handling (WSTG-ERRH-01)

INFORMATIONAL

ECC-06

Recommended browser security headers are missing

AFFECTED All responses (portal.example.test)

RETEST Still open. Scheduled for the next release. No impact on the other findings.

What we found

Responses do not set a Content-Security-Policy, and Strict-Transport-Security has a short lifetime. These headers are a second line of defense against some browser-based attacks.

Evidence

SYNTHETIC

```
HTTP/1.1 200 OK
Strict-Transport-Security: max-age=300
(no Content-Security-Policy header present)
```

Why it matters

No weakness was exploited through this. The headers would reduce the impact of a future cross-site scripting flaw.

How to fix it

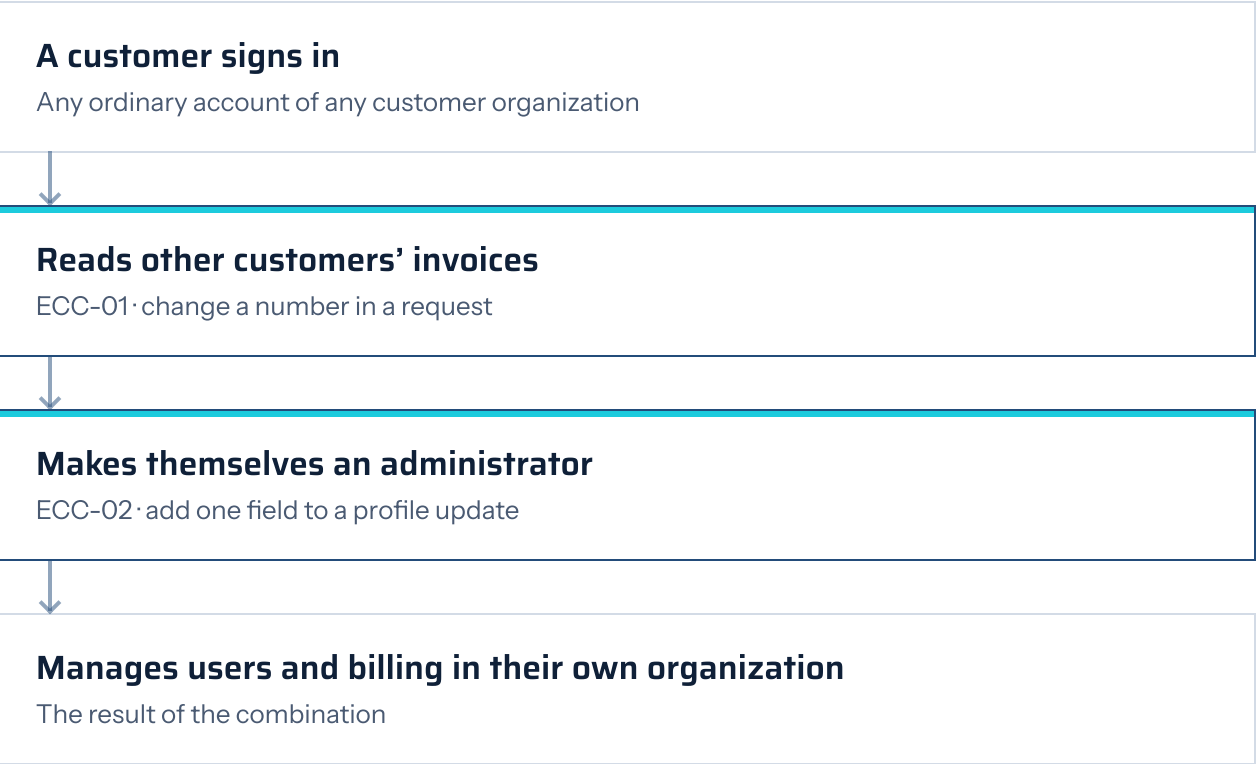
1. Add a Content-Security-Policy, starting in report-only mode, then enforce it.
2. Raise the Strict-Transport-Security lifetime once all subdomains are served over HTTPS.

References

- OWASP Secure Headers Project

CVSS scores rate the severity of a flaw in general. The report's severity also weighs how easy it is to exploit here and what the application holds.

How the high-risk findings combine



The two issues are independent. We show them together because a single account could use both. Both were fixed.

Retest results

Finding	Severity	Result at retest
ECC-01 Invoice access across organizations	High	Fixed and verified
ECC-02 Self-service role change	High	Fixed and verified
ECC-03 Account	Medium	Fixed and verified

Finding	Severity	Result at retest
enumeration on password reset		
ECC-04 Sessions stay valid after password change	Medium	Partly fixed: web sessions end, mobile tokens do not yet
ECC-05 Verbose error messages	Low	Fixed and verified
ECC-06 Missing browser security headers	Informational	Open: scheduled for the next release

The retest covers only the findings above, in the same environment, with the same accounts. It is not a new test of the whole application.

What a report like this does and does not say

It tells you

What we found, in the time and scope agreed.

- ✓ How each issue works, with evidence to reproduce it
- ✓ Which fixes come first, and why
- ✓ What changed after the retest

It does not tell you

That the system is secure.

- ✗ That nothing else is wrong. A test samples risk, it does not prove its absence
- ✗ Anything about systems outside the scope
- ✗ How the system will hold up after future changes