Securing REST API Endpoints (Against Data Leaks)

Or, How to Avoid Another ‘Optus’

Thank You to Our Sponsors and Hosts!

OWASP NEW ZEALAND
owasp.org.nz

AppSec NZ
appsec.org.nz

AUT
UNIVERSITY
TE WĀNANGA ARONUI O TAMAKI MAKAU RAU

DEFEND™

fastly®

QUANTUM SECURITY

DATACOM

aura INFORMATION SECURITY | POWERED BY KORDIA

IriusRisk

snyk

dta
Defence Technology Agency

planit an NRI company

CyberCX

tesserent

FIGHTING FOR FAIR FOR KIWI BUSINESS

safeadvisory.

Without them, this Conference couldn't happen.

Who Am I?

- Security-interested software developer (these days)
- Worked at Cosive as Security Developer from 2021 to last month
  - Currently in the market for my next role ☹
- PhD in Computer Science from University of Auckland
- Also have a BCom(Hons), spent years working in financial admin
- Overly active on InfosecNZ Discord
- Excessively reference The Simpsons

Introduction

- Focus here is specifically on preventing sensitive data from leaking out via API endpoints
  - E.g., customer or employee Personally Identifiable Information
- Limited to REST APIs/web apps only (no GraphQL, gRPC, etc.)
  - Most of it should still apply to other approaches
- Talk is not a dig at Optus—let’s learn from others’ mishaps and avoid our own data leaks!

Background

Optus Breached

- In September 2022, news came out that someone was attempting to extort major Australian telco Optus via The Dark Web™

- Claimed to have 10 million customer records scraped from Optus systems
  - That’s roughly 40% of the entire population of Australia…

- Leaked a 10,000-record subset as proof

Not-so-sophisticated Attack

- Optus CEO said it was a “very sophisticated cyber attack”.
- Australian Minister for Home Affairs Clare O’Neil was asked in a TV interview “You … don't seem to [agree] that this was a sophisticated attack?”
- The Minister responded “Well, it wasn't. **So, no**.”
- The attacker apparently found an API endpoint returning customer data with <u>no authentication requirements</u>

Lucky Country

- Intense media and government focus apparently spooked the perpetrator

- Perp took down all records and (sort of) apologised
  - Claimed no security.txt, couldn’t report vuln

- MediBank breach shortly after took focus away from Optus

- Reputational damage, ministerial derision and credit monitoring offers etc. all mean this was probably *very* expensive for Optus

Maybe not the Brightest Idea

- While this was going on, somebody started sending text messages to people from the breach
  - Threatened to expose private information if no ransom paid
- Turned out to be a Sydney teenager with no connection to the original scraping
  - Sent extortion threats to phone numbers from the leaked 10,000… Using his personal cell phone.
- Further trouble for Optus!

Some Thoughts

How Did This Happen?

- No inside knowledge, but best my guess…
- Someone turned off authentication for an endpoint during testing, and forgot to turn it back on
- Suggests organisational/structural failure
  - No relevant policies, or policies unenforced
- Nobody looking for issues/blocking bad changes

Let’s (Not) Play The Blame Game

- “Which idiot is to blame for this?”
  “Some stupid dev didn’t do their job!”
  “The reviewer should have caught it.”

- It should be very difficult for one dev to be wholly responsible.

- Implies broader organisational failings.

- Investigate the process leading to the failure, don’t search for a scapegoat.

Defensive Measures

10 not-so-weird tricks hackers don’t want you to know!

Deny by Default

- Always deny access by default.
- All unauthenticated access must be marked explicitly in code.
  - Makes it obvious if something is broadly accessible.
- All unauthenticated requests get a 401 HTTP response
  - Only exceptions are for login endpoints & related.
- “Fail secure”

To Reiterate

- Always deny by default!
- If you remember one thing from this talk, make it that
- Stops vast majority of unsophisticated attacks
- Frustrates more sophisticated attackers
- If you’re too hard to crack, they’ll probably look elsewhere

Code Reviews

- Well, duh! (hopefully)
- Reviewers should question exposed endpoints
  - (works great with ‘deny by default’)
- Try to ensure reviewers understand broader context
  - Unintended changes resulting from intended ones?
- Reviewer approval(s) mandatory
- Maybe security-focused checklists or full security reviews

Ban Changes in Production

- Evades code review/approval processes
  - Prefer CI/CD
  - Only deploy from protected branches
- Fixes often don’t get propagated
- People forget to revert temporary changes
- No guarantee someone malicious doesn’t look at that moment!
- *Probably* only in dynamic languages, but still a big problem

Control Validation Testing

- You already have lots of automated testing, right?
- Including integration & end-to-end tests?
- Just send requests to your test system and check responses
  - Unauthenticated gets 401, unauthorised gets 403, etc.
- DAST, Postman/Insomnia etc. support this
- Probably CLI tooling to do it (cURL + shell script?)

External Monitoring

- Periodic control validation testing in production
- Double-check bad changes didn’t sneak into prod
- Attempt access via the same approach as an external user
- Keep log of attempts, alert when result changes
- Monitoring should self-identify, but don’t treat it differently

Rate Limiting

- Optus leaked possibly 10 million records
- 1 record per second ≈ 16.5 weeks of requests
  - ∴ Optus’ endpoint not heavily rate limited
- All good web frameworks should have for support it
- Not always possible, but could be difference between 10,000 & 10,000,000 leaked records
- Make exceptions for certain users if needed

IP Address Allowlisting

- If users will only access from fixed origins, then only permit those origins to connect
- E.g., specific corporate networks, behind fixed IP address(es)
- Works great when different instances for different customers
- VPNs for internal-only systems (but here be dragons)
- Obviously, not always possible

OpenAPI/Swagger

- Produces a detailed listing of all endpoints
- Includes authentication requirements
- Helps make auth gaps obvious to API users
  - Ensure they can report such issues to you!
- No guarantee, but it can help
  - Enough eyeballs make shallow bugs, etc.

Security Training for Developers

- Lots of devs are self-taught these days
- Even people with CS/SE degrees don’t learn about security
- Security issues can be ‘unknown-unknowns’
  - Those are the most dangerous type
- Devs learn to spot possible problems and ask for help
- A little (awareness) training can go a long way

Penetration Tests

- Good pentesters know the ‘low-hanging fruit’
  - Raise the bar up to genuinely sophisticated attacks
- Will probably find incorrectly exposed endpoints
  - (assuming they’re in scope)
- Expensive, maybe only useful with (almost-)mature software
- Make sure you fix the underlying cause, not just the symptom!

Test vs Production

Test Environment, Production Data

- There were eventual suggestions that Optus’ leaky API endpoint was on a test environment
- Using production data source(s), however
- Maybe: “It’s just a test environment, we don’t need to worry about securing it”
- Result: Customers’ PII walks out the door. Bad time for all.

It’s Production, Unless it’s Definitely Test

- Rare not to need to worry about securing environments. Only when:
  - No sensitive data involved.
  - Environment unused/inaccessible by world outside testing
  - No ability for changes and updates made inside the environment to propagate out of it (& reset state every so often).

- If anybody not directly involved in testing will notice if it disappears, it’s not a testing environment.

- If it’s not a testing environment, all normal security measures are mandatory!

Play It Safe

When in doubt:

Treat it like it’s a production environment!

Summary

Pop Goes The Telco

- In September 2022, data of millions(?) of Optus customers was accessed by an outsider
- Apparently scraped from a REST API endpoint with no auth req.
- Optus was lucky: most data not leaked by perpetrator
  - (and the even-worse MediBank hack distracted people)
- Still serious costs in money and reputation

Do Fix Problems, Don’t Point Fingers

- Probably happened because a developer made a mistake
- Should be very difficult for that lone mistake to cause this
- Suggests larger organisational/structural issues
- Probable gaps in processes, procedures, policies or enforcement
- Blame management (if anyone), not some intern

Many Defences → Light Breaches

- No single silver bullet to stop all potential breaches
- Defence-in-depth/“Swiss cheese model”
- The more the better (usually)
- Mostly have minimal impact on performance, etc.
- Some measures technical, some cultural/structural
  - ‘Implemented’ by different people

The Top Ten

- Deny by default
- Code reviews
- No changes in production
- Control validation (automated exposure) testing
- External monitoring (‘exposure testing in prod’)
- Rate limiting
- IP address allowlisting & VPNs
- OpenAPI/Swagger
- Security training for developers
- Penetration tests

No Leaky Test Environments

- Some suggestion that Optus breach was on test environment
- Test environment connected to production data, however
  - (or pre-populated with it)
- Treat test like prod unless 100% sure
  - (deny by default strikes again!)
- May relax security if and only if
  - No sensitive/customer data
  - Changes in it can’t escape
  - Nobody outside development & test uses it

Just say no!