

IN-PERSON WORKSHOP FOR TECHNICAL TEAMS

Secure development

Find and fix the security flaws that most often reach production in web apps and APIs

One day for the team to exploit, fix and test the flaws that keep showing up in web applications and APIs: broken authorisation, injection, leaked secrets and dependencies nobody looks at. We spend the last block hunting for them in the company's own code.

5 to 8 h

on-site, depending on the team

6

hands-on labs

6

content blocks

1

block on the company's code

WHERE TEAMS START

The team knows the OWASP Top 10 and the code still has the same flaws

Most developers have seen the OWASP list at some point. Still, the first category in the 2025 edition is broken access control. The team knows it. What's missing is checking on the server that the requested object belongs to the person asking for it.

Meanwhile the risk has moved to code the team didn't write: packages that run their own install scripts, CI actions referenced by a tag anyone with access can rewrite, and tokens that still work years after they leaked.

In this workshop we don't walk through the list. Each participant exploits the flaw, tries the fix that looks obvious, sees it isn't enough, and leaves a test that proves the real fix works. Then they run the same search on their own code.

The five places where an application breaks

01 Design

What can go wrong, sketched on one page before anyone writes code.

02 Authorisation

Who can read or change each object, checked on the server.

03 Inputs

Where untrusted data gets in and what it ends up as: SQL, HTML, a URL or an object.

04 Identity and secrets

Tokens, sessions, credentials and whatever ends up in the logs.

05 Supply chain and CI

Dependencies, lockfiles, pipelines and the gates that block the merge.

Five areas, with a tested control in each one. The team takes them back to its repository when the day is over.

WHAT THE TEAM GETS

Five things the team can do by the end of the day

Each one is shown during the day with an exploit, a fix and a test.

01

Threat model a flow in fifteen minutes

Answer the four questions from the Threat Modeling Manifesto for a real flow and mark the trust boundaries before arguing about controls.

Evidence: a one-page sheet for the checkout of the practice application.

02

Lock down object and property authorisation

Tell BOLA, BFLA and mass assignment apart, and enforce the check in the data layer instead of the UI.

Evidence: the exploit that read other customers' orders now gets a 404, and a test pins it.

03

Handle each input according to where it ends up

Parameterised queries, context-aware escaping, a destination allowlist for SSRF, and no pickle or yaml.load on outside data.

Evidence: the lab's injection and SSRF payloads stop working.

04

Keep secrets out of the code and out of the logs

Validate tokens against a fixed list of algorithms, take secrets out of the repository, rotate what leaked and clean up what gets logged.

Evidence: an alg=none token rejected and a log with no passwords or card data.

05

Set up CI gates that actually block

Lockfiles with hashes, SBOM, SAST, SCA, secret scanning and actions pinned by SHA, with a clear decision on what breaks the build.

Evidence: a malicious pull request against the pipeline is stopped before it runs with secrets.

REAL CASES

2025 incidents explained by what we practise that day

All of them are public, documented by whoever found them or by the vendor. None needed an exotic technique.

BROKEN OBJECT LEVEL AUTHORIZATION**McHire exposed more than 64 million job applications**

A test account with 123456 as both username and password opened the admin panel. From there, changing a sequential ID in the request returned other applicants' data. It was reported on 30 June 2025 and fixed the next day.

CONTROL PRACTISED

Ownership check on every query, unpredictable identifiers and no test accounts left in production.

INSECURE DESIGN**Bypassing Next.js middleware with one header**

In March 2025, CVE-2025-29927 let attackers skip the middleware by sending the x-middleware-subrequest header. Applications that only checked authorisation there were left open.

CONTROL PRACTISED

Authorise next to the data as well, and don't rely on a single layer an attacker can route around.

INSECURE DESERIALIZATION

React2Shell: unauthenticated remote code execution, CVSS 10.0

CVE-2025-55182, published on 3 December 2025, allowed code execution on the server through deserialization in React Server Components.

CONTROL PRACTISED

A dependency inventory that shows within hours which services are affected, and no deserialization of object formats from external input.

SOFTWARE SUPPLY CHAIN FAILURES

Shai-Hulud 2.0 ran before the tests did

Between 21 and 24 November 2025, compromised npm packages shipped preinstall code that stole cloud credentials and npm and GitHub tokens the moment they were installed.

CONTROL PRACTISED

npm ci with a lockfile, install scripts disabled in CI, and publishing protected with trusted publishing and WebAuthn.

CI/CD

tj-actions/changed-files dumped secrets into thousands of build logs

In March 2025 someone rewrote the action's tags to point at a malicious commit. More than 23,000 repositories used it, and the ones referencing it by tag ran the new code without touching anything.

CONTROL PRACTISED

Actions and tasks pinned by full SHA, minimal CI token permissions and no secrets exposed to external pull requests.

What I teach, block by block

The order follows a request: the flow gets designed, someone asks for an object, sends data, authenticates, and the change goes through the pipeline before it reaches production. The categories come from the OWASP Top 10:2025 and the OWASP API Security Top 10 2023, and each control maps to its ASVS 5.0 requirement.

01 Lightweight threat modelling

Before talking about controls, we need to know what we are protecting and from whom.

- The four questions: what are we working on, what can go wrong, what are we going to do about it, did we do a good enough job.
- Trust boundaries: browser, API, database, third-party services and cloud metadata.
- Who may do this, where untrusted data gets in, and how far the damage reaches.
- Using ASVS 5.0 as a list of requirements and not as background reading.

Real case: CVE-2025-29927 in Next.js: authorisation lived in a single layer.

02 Authorisation: number one in the Top 10

Being authenticated says nothing about what a user may touch.

- BOLA and IDOR: reading or changing someone else's objects by changing an ID.
- BFLA and BOPLA: exposed admin functions and properties the client should never be able to write.
- Mass assignment: the role field or the price arriving in the request body.
- Sensitive business flows: amounts calculated on the client and steps that can be skipped.

Real case: McHire, 2025: more than 64 million applications reachable by changing a number.

03 Injection and SSRF

The same piece of data is harmless or dangerous depending on where it lands.

- SQL and NoSQL: why hand-rolled escaping fails and what a parameterised query does.
- Stored XSS and the token kept in localStorage.
- SSRF: a link preview that ends up asking the cloud metadata service for credentials.
- Destination allowlists, DNS resolution and redirects, which is where text-based filters break.

04 Authentication, tokens and deserialization

A signed token only protects anything if the server decides how it gets validated.

- JWT: alg=none, algorithm confusion and the fixed list that prevents both.
- Sessions, expiry, revocation and where to keep the token in the browser.
- Deserialization: pickle, yaml.load and any format that rebuilds objects.
- Authentication failures: default passwords, test accounts and rate limits on attempts.

Real case: React2Shell, CVE-2025-55182: unauthenticated remote code execution through deserialization.

05 Secrets, sensitive data, logs and errors

What leaks tends to sit in plain text somewhere nobody looks.

- Secrets in code and in git history: detection, removal and rotation.
- Card and personal data at rest: what never gets stored and what gets encrypted.
- Logs that record passwords and error responses that show the stack trace.
- Debug mode in production and handling exceptional conditions (A10:2025).

Real case: GitGuardian, 2026: 28.65 million new secrets in public GitHub commits during 2025, and over 64% of credentials valid in 2022 still worked in January 2026.

06 Supply chain and CI gates

Third place in the Top 10:2025, and almost all of it is code the team didn't write.

- Lockfiles and hashes: npm ci, pip --require-hashes and install scripts turned off.
- SBOM in CycloneDX or SPDX, image scanning and deploying by digest.
- Pipelines: pull requests that edit the YAML, dependency confusion and actions pinned by SHA.
- SAST, DAST, SCA and secret scanning: what each one finds and which ones block the merge.

Real case: Shai-Hulud 2.0 on npm, tj-actions/changed-files on GitHub Actions and num2words on PyPI, all in 2025.

DELIBERATE PRACTICE

Six labs on a vulnerable online shop

The labs use ByteBazaar, a Flask shop backed by PostgreSQL and MongoDB that runs with Docker Compose. It's public and open source (github.com/cr0hn/vulnerable-python). Every lab follows the same pattern: exploit it, try the fix that looks obvious, see why it fails, apply the real one and leave a test behind.

01

Another customer's orders

With a normal account, each participant reads other people's orders by changing the ID in the URL. Then they try hiding the link in the UI and find out it changes nothing.

Outcome: the query filters by owner, and a test fails if anyone removes the filter.

02

Becoming admin and paying any price

Sign-up accepts a role field sent by the client, and checkout trusts the amount coming from the browser.

Outcome: an explicit list of writable fields and the amount recalculated on the server.

03

Searches that return the whole database

SQL injection in the product search and NoSQL injection in the tag filter. The first fix escapes quotes by hand and falls again.

Outcome: parameterised queries in both databases and the payloads kept as test cases.

04

The link preview that asks for the cloud keys

A feature that previews links gets used to query the simulated metadata service and pull credentials from a secrets store.

Outcome: destination allowlist, validated resolution and redirects blocked.

05

An unsigned token and logs that tell everything

We forge an alg=none JWT to log in as another user. Then we read the application logs and find passwords and card data.

Outcome: allowed algorithms fixed on the server and logs that no longer record sensitive data.

06

The pipeline that can be fooled

A pull request edits the pipeline YAML to read secrets, and an internal dependency gets spoofed from the public registry. We compare the insecure Azure DevOps pipeline with its hardened version.

Outcome: SCA, SAST and secret scanning gates, dependencies resolved from the right registry and secrets kept away from pull requests.

ON THE COMPANY'S CODE

We do the last part on the company's own code

What we practised on ByteBazaar, each participant then looks for in their own code. That way the team ends the day with findings of its own as well as examples from another application.

01

Pick a flow that matters

One that moves money or personal data. We run the fifteen-minute threat model on it with the four questions.

02

Look for the day's patterns

Objects loaded by ID without filtering by user, concatenated SQL, requests to URLs supplied by the client, pickle or yaml.load on outside data. A secret scanner also goes through the git history.

03

Check the supply chain

Whether CI installs from the lockfile, whether actions or tasks are pinned, and which scripts run when dependencies are installed.

04

Write a failing test

For the most serious finding, a test that fails today and passes once it's fixed.

05

Decide the first action

Each participant says out loud what they'll fix first and why. If they don't want to, they don't show the code.

Each participant works on their own laptop, in their own repository. The code never leaves the company and I don't take a copy. It's not an audit: the findings belong to the team and stay with it.

Between 5 and 8 hours, depending on the team

We alternate short explanations with practice. Every block ends with a tested control, and the last one is entirely for the company's code.

Block 1

Design before attacking

Lightweight threat model of the ByteBazaar checkout. We start the application with Docker and mark the trust boundaries.

Block 2

Who can touch what

BOLA, BFLA, mass assignment and business flows. Labs 1 and 2.

Block 3

Data that turns into code

SQL and NoSQL injection, XSS and SSRF against cloud metadata. Labs 3 and 4.

Block 4

Identity, secrets and what stays in the logs

JWT, sessions, deserialization, secrets in git and error handling. Lab 5.

Block 5

What the team doesn't write

Dependencies, lockfiles, SBOM, real attacks on npm and PyPI, and CI gates. Lab 6.

Block 6

The company's code

Each participant applies the day to their own repository: threat model, pattern search, supply chain review and a failing test. We close with everyone's first action.

MATERIAL THAT STAYS WITH THE COMPANY

The workshop ends and the controls stay

The material is made to be used in the repository and in code reviews over the following weeks.

One-page threat model sheet

The four questions, trust boundaries and a worked example.

Control map

OWASP Top 10:2025 and API Security Top 10 mapped to ASVS 5.0 requirements and to each lab.

Reference pipeline

Azure DevOps with the day's gates. The same controls carry over to GitHub Actions or GitLab CI.

Application and labs

ByteBazaar and the lab briefs, to repeat the exercises or use them with new team members.

Per-endpoint authorisation checklist

Object, function and property, with the test that goes with each case.

Fix pairs

The fix that looks right, why it fails and the one that works, for every vulnerability of the day.

Dependency policy

Lockfiles, hashes, install scripts, SBOM and what to do when a well-known package turns out to be compromised.

WHAT COMES WITH THE COURSE

Four things that stay with the team, at no extra cost

They're what teams end up asking for after a course.

+1

Compliance evidence pack

A signed attendance sheet, an individual certificate with date, hours and contents, a syllabus mapped to the standards that apply to the company, and a one-page letter for the auditor.

+2

A 10-minute survey beforehand

Before the day, the team answers a short survey about its stack and where it gets stuck. I use it to adapt the labs to what the team really uses.

+3

Configurations on a branch of the company's repository

During the final block I leave the configurations we tested on a branch: regression tests, pipeline checks and merge rules. The team reviews and merges them like any other change.

+4

Labs with solutions

The lab repository with its solutions and a 12-month internal-use licence, so new people on the team don't depend on repeating the day.

COMPLIANCE

What each standard asks for and what I hand over as evidence

The course doesn't make anyone compliant by itself. It works as documented evidence for the training control the auditor already asks about, if the standard applies to the company.

STANDARD	WHAT IT ASKS FOR	WHAT I LEAVE AS EVIDENCE
PCI DSS 4.0 · 6.2.2	Training at least every 12 months for people who develop bespoke software: secure design, secure coding and testing tools, documented.	Syllabus mapped to those points, signed attendance and date.
ISO 27001:2022 · A.6.3	Security awareness, education and training appropriate to each person's role.	Individual certificate with date, hours and contents, and a per-person record.
ISO 27001:2022 · A.8.25 and A.8.28	Secure development life cycle and secure coding, including training for those who develop.	Mapped syllabus and the pipeline checks that stay on the company's branch.
SOC 2 · CC1.4 and CC2.2	A documented training programme, with completion records per person and depth matched to the role.	List with name, date, contents and role.
DORA · art. 13(6)	Digital operational resilience training for staff, proportionate to their role (banking and insurance).	Attendance record and course syllabus.

For teams that run web applications and APIs in production

A strong fit for

- Backend and full-stack developers who write endpoints, queries and business logic.
- Tech leads who review pull requests and decide what blocks a merge.
- Platform or DevOps teams that run the pipeline and the dependencies.
- Security champions who will take these practices to the rest of the organisation.
- Teams on any stack: the labs are in Python, but the controls are the same in other languages.

Not the right workshop if

- The team doesn't write or maintain application code.
- The company is looking for a security introduction for non-technical staff or leadership.
- What is needed is an audit or a pen-test of the applications. The team does the final block on its own code, and it doesn't replace an external review.
- The goal is to prepare for a specific certification.

Security and development explained by someone who has worked on both sides



I'm Daniel Alfocea, and I've spent twenty years in development and security. At Banco Santander I was Principal Engineer in the AI department while the area was being set up. At BBVA I was Cybersecurity Tech Lead. At 42Crunch, in the UK, I was responsible for one of its API security products.

I've presented research at RSA Conference and OWASP AppSec EU, and I'm a regular speaker at RootedCON. Spain's National Cybersecurity Institute (INCIBE) has asked me to judge its competitions several times. I founded Navaja Negra and the OWASP Madrid chapter. I'm the author of widely used security tools such as GoLismero, Plecost and Dockerscan, and of LAPIS, research published on arXiv on how to describe an API to a model with fewer tokens. With Alfonso Muñoz I wrote *Secure MCP: A practical guide for developers, software architects and tech leads*.

The lab application is mine and it's public. It gathers the flaws any development team can write in an ordinary week, and in the workshop each one gets fixed with a test that stops it from coming back.

GUARANTEES

Two commitments in writing and one limit

If the configurations don't work, I fix them

If the configurations I leave in the company's repository don't work, I run a free remote session to fix them.

Stuck after 30 days, one free hour

If after 30 days the team has got stuck applying something from the course, I spend an hour with the team remotely, free.

This isn't an audit or a promise of results: I don't guarantee percentage improvements or that there will be no incidents.

Questions worth settling now

How much does it cost?

It depends on the duration, between five and eight hours, and on the size of the team. After the first conversation I send a fixed proposal.

How is this different from the secure development with AI workshop?

This one doesn't cover coding assistants. It's about application and supply chain vulnerabilities, whether a person or a tool wrote the code. If the team already works daily with Claude Code or GitHub Copilot, the other workshop covers the risks specific to those tools.

Does it adapt to our language or stack?

The labs are in Python and the pipeline example uses Azure DevOps. The controls are the same in other languages and CI systems: authorise next to the data, parameterise queries, pin dependencies and block the merge without evidence. I don't rewrite the labs in another language.

Do we use our own repository during the day?

Usually yes, in the last block. The labs run on ByteBazaar because it's built to break without consequences and the whole group sees the same failure. After that, each participant applies what they learned to their own code, on their own laptop. If the company can't open its code that day, we work on a practice repository similar to its stack.

Do you see or take our code?

No. The code doesn't leave the company and I don't take a copy. If a participant wants to show me something, I'll help with the question, but it's not an audit and there's no findings report.

What level is needed?

Participants need to read basic Python, use git and run containers with Docker. No previous security experience is required.

Is there a certificate?

Yes. Each person gets an individual attendance certificate with date, hours and contents. There's no exam, so it proves attendance and isn't a qualification.

Can it be delivered remotely?

Yes, although in person works better for unblocking labs and helping in the block on the company's code. The scope is the same.

Does it guarantee we'll have no security incidents?

No. What I guarantee is the six labs and the material. Whether the risk goes down depends on the team applying and maintaining the controls afterwards.

What if we need follow-up?

There's a 6-week programme on this same topic, with measurement before and after and review of the team's PRs. The details are on [the programme page](#).

FORMAT

One team, one room and the same practice code

It covers preparation, between five and eight in-person hours depending on the team, and all the working material. With small groups I can review the labs while people work on them and help each participant during the block on their own code. With a crowded room, practice turns into a demo.

- Participants from the same team or from areas that work together.
- I run it in English or Spanish.
- Room, screen, wifi and laptops provided by the company, with Docker installed and access to each participant's repository.
- Travel outside Madrid agreed before the date is confirmed.
- It includes the evidence pack, the configurations on the company's branch and the two commitments above.
- It is designed as a working day for a team that will apply what it learns. It is not an audit, a compliance certificate or a talk to sit through.

Write to me at

me@danielalfocea.com

In the first conversation we check whether the workshop is a fit, how many people will take part and which date works for the company.

Can be subsidised through FUNDAE (Spanish training credit).

EMAIL

me@danielalfocea.com

PROGRAMME ONLINE

danielalfocea.com/en/formacion/workshop-desarrollo-seguro