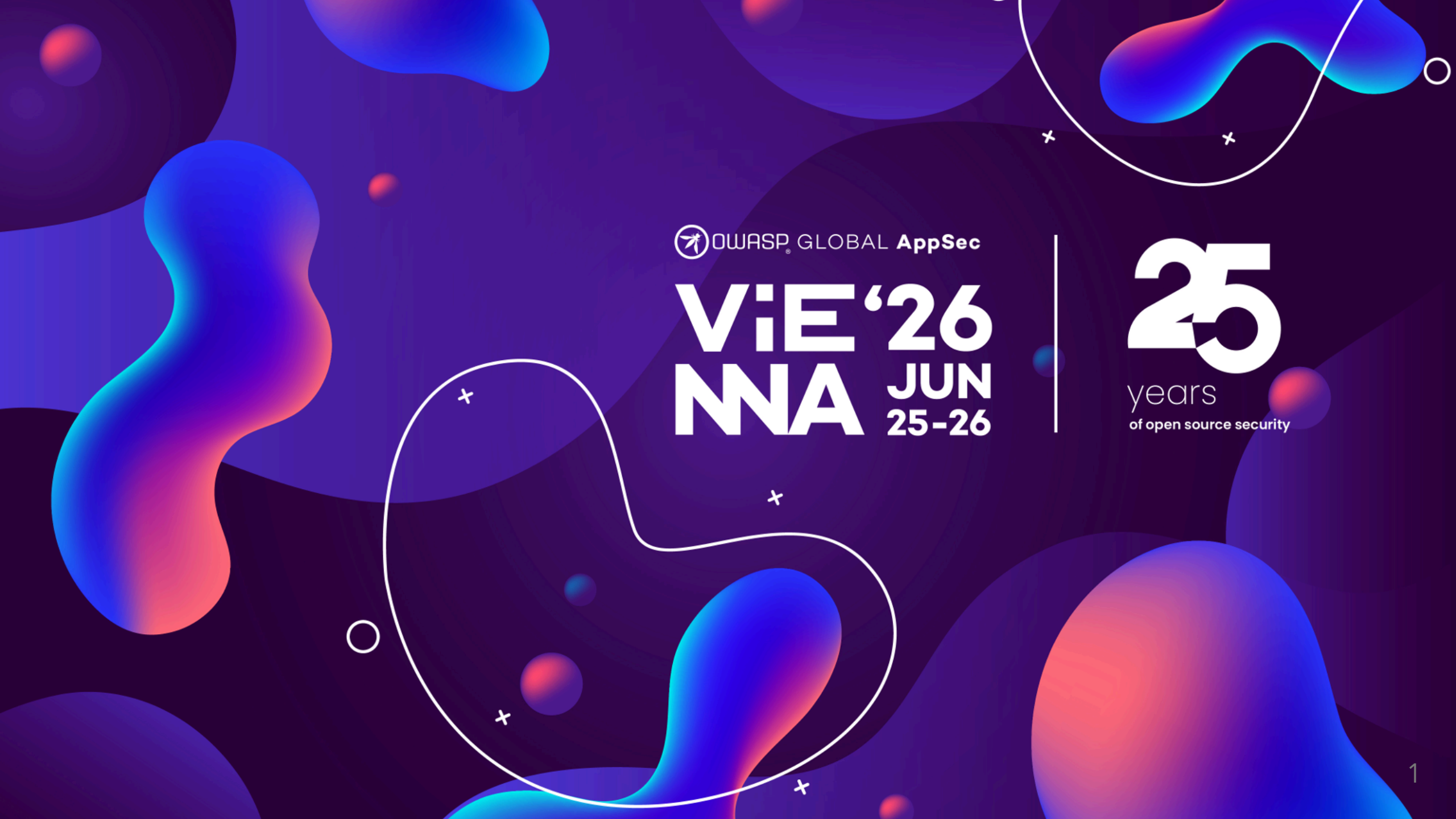
OWASP GLOBAL AppSec
VIE '26
NNA JUN
25-26
25
years
of open source security

OWASP GLOBAL AppSec

VIENNA '26 JUN 25-26

25 years of open source security

Why AppSec Fails at Scale and how to fix it

by: Eduard Thamm

The Pattern We Know

A vulnerability is found.

It is triaged, ticketed, fixed, verified, and closed.

Three months later, the same class of vulnerability appears somewhere else.

That is not just "a finding"

> A recurring vulnerability is a signal that the system learned nothing.

What we usually do

More tools

More scanners, dashboards, alerts, integrations.

More rules

More policies, checklists, guidelines, mandatory steps.

More pressure

More chasing, escalation, blame.

The result

Security feels busier

- More findings
- More meetings
- More exceptions
- More reporting

But risk does not move much

- Vulnerabilities recur
- Teams route around process
- Security becomes late friction
- Trust erodes

AppSec fails at scale when we manage findings instead of designing systems that make secure behavior the easiest path.

Part 1

Why AppSec breaks as organizations grow

Scaling changes the problem

Early on, AppSec can survive on proximity.

- Everyone knows the system
- People ask the security person directly
- Reviews happen through relationships
- Exceptions are remembered informally

At scale, proximity disappears.

The Shape of AppSec

Security outcomes are shaped by:

```
Defaults → Workflows → Choices → Outcomes → Feedback
    ↑           ↓↑         ↑
Ownership  Incentives  Metrics
```

Security outcomes are produced by the system around the engineer.

Pattern 1

Compliance-driven security

> Compliance asks: "Can we prove we satisfy this control?"
> Security asks: "Did the control reduce the risk?"

Mature organizations need both perspectives.

Pattern 2

Dashboard-thinking

We count what tools emit. Tools emit numbers. Numbers become targets.

- Critical (by some definition) findings
- SLA compliance
- Training completion
- Scan coverage
- Open ticket count

Pattern 3

Ticket-driven AppSec

A finding becomes a ticket.

A ticket becomes work in some backlog.

Work follows flow to closed state.

Something around ticket closure becomes a metric.

But the system that produced the finding remains unchanged.

Ticket closure is not learning

Ask after every repeated finding:

- Was there an unsafe default?
- Was the safe path hard to discover?
- Was ownership unclear?
- Was feedback too late?
- Did delivery pressure make the insecure choice rational?

Pattern 4

"Shift left" as responsibility dumping

Shift-left fails when it moves accountability without moving capability.

Engineers get more checklists, more tools, more findings — but not better defaults, better APIs, or better paths.

Security is everyone's responsibility?

Yes.

And when everyone is responsible and nobody owns the mechanism, outcomes depend on luck, attention, heroics, and other random factors.

The force underneath all of this

Delivery Pressure

Security advice that ignores delivery pressure will be routed around.

Not because engineers are careless.

Because the system rewards shipping (fast and often).

The underlying constraint

If secure work is slower, harder, and less rewarded, insecure work will keep winning more often than not.

Part 2

What could the system look like?

The Central Question

Why was this easy to do insecurely?

Example: secrets in logs

Why was it easy to leak secrets into logs?

- The default logger accepts arbitrary fields
- Examples show full request/response dumps
- Review caught it late/after go-live
- Teams had no safe logging lib/helper
- Ownership sat with product teams, but the fix belonged in the platform

The Mechanism Loop

Recurring finding class → Systems diagnosis → Secure default / paved road / other action → Adoption signal → Recurrence drops

Example: secrets in logs → safe logging wrapper → CI guardrail → services adopt → fewer repeated findings

From findings to mechanisms

Non-Scaling pattern

"Please remember to handle this securely."

Scalable pattern

"The platform makes the secure behavior the default."

Mechanism 1

Secure defaults

Defaults are security decisions made once and reused many times.

Examples:

- Safe HTTP headers in the framework
- Authentication middleware by default
- Secure cookie/session settings
- Standardized logging without secrets
- Safe dependency update behavior

Secure defaults reduce cognitive load

The goal is not to make every engineer remember every security decision.

The goal is to remove boring security decisions from everyday delivery.

Mechanism 2

Golden paths / paved roads

A paved road earns adoption.

It is the fastest supported way to do common work safely.

What can be turned into paved road?

Identity

Auth, sessions, authorization boundaries.

Delivery

CI/CD, signing, deployment guardrails.

Data

Storage, logging, retention, secrets.

Mechanism 3

Platform-owned controls

Some controls should not be re-implemented by every team.

- Base images
- Build templates
- Secret delivery
- Runtime policy
- Observability defaults
- Dependency update automation

Platform ownership changes the conversation

Before

"Team A forgot the hardening guideline."

After

"The platform template did not make hardening automatic."

Mechanism 4

Fast feedback loops

Security feedback should arrive while the engineer still has context.

Good feedback is:

- Early
- Specific
- Actionable
- Low-noise
- Connected to ownership

Late feedback creates conflict

A finding after implementation feels like rejection.

A constraint during design feels like engineering input.

Designing the mechanism

- Safe defaults
- Reusable paths
- Risk-based exceptions
- Fast feedback
- Clear ownership
- Signals that steer decisions

Every system fails

This model has failure modes as well.

- Quiet bypassing of secure defaults
- Slow paved roads abandoned
- Platform controls becoming single points of failure
- ...

This Does Not Replace Your Compliance Program

Secure defaults and platform controls reduce the manual work that audits surface as findings.

- Recurring finding class eliminated → audit evidence becomes lighter
- Platform-owned controls → evidence collection becomes easier
- Fewer exceptions → exception tracking becomes manageable

The argument is not: stop doing compliance.
It is: stop treating compliance as the ceiling.

Part 3

How to measure progress that means something

Metrics are steering signals

Metrics are not truth.

They are instruments that help you notice whether the system is moving in the right direction.

Activity metrics are easy

- Number of scans
- Number of findings
- Number of closed tickets
- Number of trained engineers
- Number of policy exceptions

Better Signals

Signal	Tells you about...
Recurrence by vulnerability class	whether the system learns
Time from introduction to detection	feedback being early/late
Time from detection to remediation	fix complexity, ownership clarity, delivery pressure
Time from detection to acceptance	decision chain performance
Services on secure defaults	whether platform mechanisms spread
Exception age and review rate	risk 'maintenance'

For our Example

Better signals are not:

- How many log findings did we close?

Better signals are:

- How many services use the safe logger?
- How many still bypass it?
- Are new secrets-in-logs findings decreasing?
- Are exceptions expiring or becoming permanent?

A Cornerstone

How many security decisions did we remove from product teams?

For example:

- Auth is handled by middleware now
- TLS, headers and logging for new services are done automatically

A practical adoption path

Short Term

Pick one recurring finding class. Map why it recurs.

Mid Term

Build or improve one secure default or paved path.

Long Term

Measure recurrence and adoption. Retire one manual review/interaction.

Choose boring wins first

Potentially good starting points:

- Auth middleware defaults
- Secret handling patterns
- Secure service templates
- CI checks with clear fixes
- Dependency update automation
- Exception ownership and expiry

The mindset shift

Less

"Who failed to follow the process?"

More

"What made the risky choice reasonable?"

What to do on Monday

If you review code	If you lead across teams	If you fund security
Turn repeated comments into checks/templates	Pick one recurring vuln class and build a paved road	Fund mechanisms, not just finding management

The Key

Scalable AppSec is not about making every engineer a security expert.

It is about building an engineering system where secure delivery is normal, fast, and repeatable.

The Work

The work is not to find more problems. The work is to build systems that stop producing the same problems.


OWASP GLOBAL AppSec
VIENNA'26 JUN 25-26
25 years
of open source security
THANK YOU!